

AD-A173 442

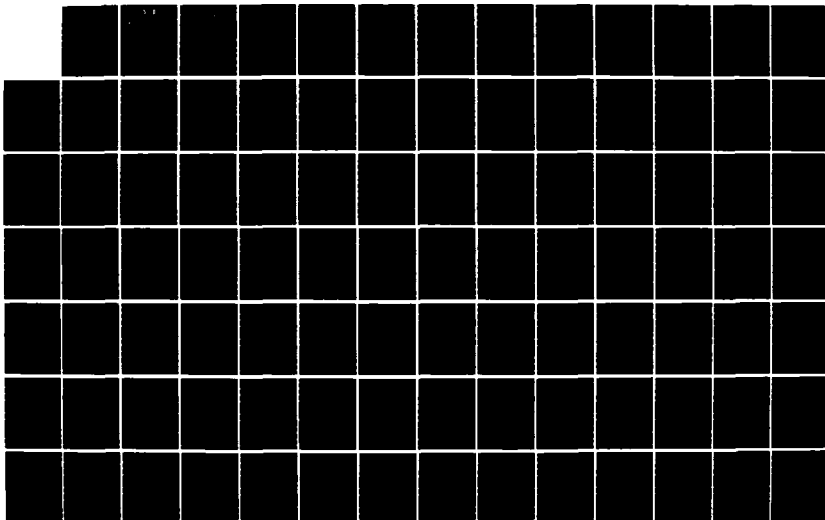
DEVELOPMENT OF A VALIDATION SYSTEM FOR GROUP 4  
FACSIMILE EQUIPMENT(U) DELTA INFORMATION SYSTEMS INC  
HORSHAM PA NOV 85 NCS-TIB-85-8 DCA100-83-C-0047

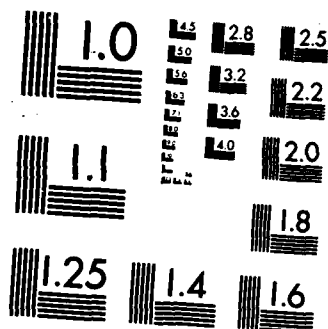
1/2

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

DTIC  
ELECTE  
OCT 27 1986  
S D

(2)

NCS TIB 85-8



## NATIONAL COMMUNICATIONS SYSTEM

AD-A173 442

### TECHNICAL INFORMATION BULLETIN 85-8

### DEVELOPMENT OF A VALIDATION SYSTEM FOR GROUP 4 FACSIMILE EQUIPMENT

NOVEMBER 1985

DTIC FILE COPY

DISTRIBUTION STATEMENT A  
Approved for public release;  
Distribution Unlimited

86 10 013

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

A173442-

## REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE			<b>DISTRIBUTION STATEMENT A</b> Approved for public release Distribution Unlimited		
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NCS TIB 85-8					
5. MONITORING ORGANIZATION REPORT NUMBER(S)					
6a. NAME OF PERFORMING ORGANIZATION Delta Information Systems		6b. OFFICE SYMBOL (If applicable)		7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State, and ZIP Code) Horsham Business Center Bldg. 3 300 Welsh Road, Horsham, PA 19044		7b. ADDRESS (City, State, and ZIP Code)			
8a. NAME OF FUNDING / SPONSORING ORGANIZATION National Communications System		8b. OFFICE SYMBOL (If applicable) NCS-TS		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DCA 100-83-C-0047	
8c. ADDRESS (City, State, and ZIP Code) Technology & Standards Washington, D.C. 20305-2010		10. SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO.		PROJECT NO.	TASK NO.
					WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Development of a Validation System for Group 4 Facsimile Equipment					
12. PERSONAL AUTHOR(S)					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Year, Month, Day) Nov 1985	
15. PAGE COUNT 99					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Group 4 Facsimile		
			Public Switched Telephone Network (PSTN)		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The report is divided into three principal areas: a system overview, system design and system operation are presented. The system overview reviews the overall validation system requirements and the system structure used to meet these requirements. It also outlines the software and hardware design taken to satisfy the CCITT Recommendations governing the Teletex Protocol structure for Group 4 Facsimile equipment. The system design, Section 3, discusses in detail the system design. It also reviews the hardware and software design approaches for implementing the Teletex protocol structure. This structure is consistent with the seven-layer Open Systems Interconnection (OSI) reference model, which provides a basic framework for protocol development. The system operation, Section 3, details the operation of the validation system and explains the parameterization required to define the protocol variants for the testing of a Group 4 terminal.					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED / UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Dennis Bodson			22b. TELEPHONE (Include Area Code) 202-692-2124		22c. OFFICE SYMBOL NCS-TS

NCS TECHNICAL INFORMATION BULLETIN 85-8

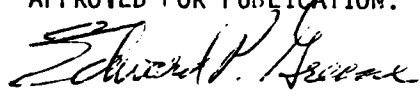
DEVELOPMENT OF A VALIDATION SYSTEM  
FOR GROUP 4 FACSIMILE EQUIPMENT

NOVEMBER 1985

PROJECT OFFICER

DENNIS BODSON  
Senior Electronics Engineer  
Office of NCS Technology  
and Standards

APPROVED FOR PUBLICATION:

  
for MARSHALL L. CAIN  
Assistant Manager  
Office of NCS Technology  
and Standards

FOREWORD

Among the responsibilities assigned to the Office of the Manager, National Communications System, is the management of the Federal Telecommunication Standards Program. Under this program, the NCS, with the assistance of the Federal Telecommunication Standards Committee identifies, develops, and coordinates proposed Federal Standards which either contribute to the interoperability of functionally similar Federal telecommunication systems or to the achievement of a compatible and efficient interface between computer and telecommunication systems. In developing and coordinating these standards, a considerable amount of effort is expended in initiating and pursuing joint standards development efforts with appropriate technical committees of the Electronic Industries Association, the American National Standards Institute, the International Organization for Standardization, and the International Telegraph and Telephone Consultative Committee of the International Telecommunication Union. This Technical Information Bulletin presents an overview of an effort which is contributing to the development of compatible Federal, national, and international standards in the area of facsimile standards. It has been prepared to inform interested Federal activities of the progress of these efforts. Any comments, inputs or statements of requirements which could assist in the advancement of this work are welcome and should be addressed to:

Office of the Manager  
National Communications System  
ATTN: NCS-TS  
Washington, DC 20305  
(202) 692-2124

DEVELOPMENT OF A VALIDATION SYSTEM  
FOR GROUP 4 FACSIMILE EQUIPMENT

November, 1985

Final Report

Submitted to:

NATIONAL COMMUNICATIONS SYSTEM  
Office of Technology and Standards  
Washington, DC 20305

Contracting Agency:

DEFENSE COMMUNICATIONS AGENCY

Contract Number - DCA100-83-C-0047



Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By <i>per form 50</i>	
Distribution	
Availability Codes	
Dist	Avail and/or Special
<i>A-1</i>	

## Table of Contents

<u>Section</u>	<u>Page</u>
1.0 Introduction. . . . .	1- 1
2.0 Group 4 Validation System Overview. . . . .	2- 1
3.0 Validation System Design. . . . .	3- 1
3.1 Software Design Approach. . . . .	3- 1
3.1.1 System Concept . . . . .	3- 1
3.1.2 Validation System Description. . . . .	3- 5
3.2 Hardware Design Approach. . . . .	3-13
3.2.1 Overview . . . . .	3-13
3.2.2 PDI Characteristics. . . . .	3-13
3.2.3 DEC 11/60 Parallel Data Interface. . . . .	3-14
3.2.4 HP 1000 Parallel Data Interface. . . . .	3-14
4.0 Validation System Operation . . . . .	4- 1
4.1 Test Selection. . . . .	4- 1
4.2 Test Validation . . . . .	4- 4
4.3 Test Configuration. . . . .	4- 4
4.4 Test Execution. . . . .	4- 9
4.5 Test Post-mortems . . . . .	4-11
APPENDIX A - Validation System Software Description	
APPENDIX B - Validation System Hardware Description	
APPENDIX C - Test Selection and Processing Syntax	

## 1.0 Introduction

This document summarizes work performed by Delta Information Systems, Inc., for the Office of Technology and Standards of the National Communications System, an organization of the U. S. Government, headed by National Communications System Assistant Manager Marshall L. Cain. Mr. Cain is responsible for the management of the Federal Telecommunications Standards Program, which develops telecommunications standards, the use of which are mandatory by all Federal agencies. The purpose of the initial task, performed under contract number DCA100-83-C-0047, was the development of the software required to validate Group 4 Facsimile equipment over a Public Switched Telephone Network (PSTN). Subsequent to the initiation of the first task, a second task which expanded the Group 4 Validation System to include operation over a Packet Switched Public Data Network (PSPDN) was added under Task Order Number 84-003 of Modification P00004, contract number DCA100-83-C-0047.

In the following three sections a system overview, the system design and the system operation are presented. Section 2 reviews the overall Validation system requirements and the system structure used to meet these requirements. It also outlines the software and hardware design taken to satisfy the CCITT Recommendations governing the Teletex Protocol structure for Group 4 Facsimile equipment. Section 3 discusses in detail the system design. It reviews the hardware and software design approaches for implementing the Teletex protocol structure. This structure is consistent with the seven-layer Open Systems Interconnection (OSI) reference model, which provides a



basic framework for protocol development. Section 4 details the operation of the validation system and explains the parameterization required to define the protocol variants for the testing of a Group 4 terminal.

## 2.0 Group 4 Validation System Overview

The primary purpose of the Group 4 Validation system is the testing/evaluation of a Group 4 Facsimile terminal. Shown in Figure 2-1 is a simplified block diagram of the validation system architecture. The function of the validation system is to insure that the Group 4 terminal (UUT) has properly implemented the protocols required for layers 3 through 6 of the Teletex Protocol structure for Group 4 Facsimile equipment and also conforms to the allowable parameter variations (e.g. buffer sizes, timeout periods, etc.) within each protocol layer. Since layer 7, the Application layer, currently has no defined protocol this layer cannot be validated. In addition protocol testing of layers 1 & 2 (Physical layer & Link layer) is not performed. However, any protocol violations or nonrecoverable errors are reported to the higher layers.

Shown in Figure 2-2 is a diagram of the Group 4 Validation system structure. The validation system was implemented with layers 3 through 7 and the necessary control routines (e.g. test control, error logging, etc.) using Delta Information System's HP 1000 processor. As seen in Figure 2-2, layers 1 & 2 were implemented using a microprocessor controlled Packet Data Interface (PDI) board developed by Delta Information Systems. The PDI interfaces with the HP 1000 bus and to the appropriate modem for either a Public Switched Telephone Network (PSTN) or a Packet Switched Public Data Network (PSPDN).

The Group 4 Validation System software consists of the system executive or test control side, the Group 4 terminal emulator and a

# GROUP 4 VALIDATION SYSTEM

GROUP 4 TEST GENERATOR/G4 TERMINAL EMULATOR

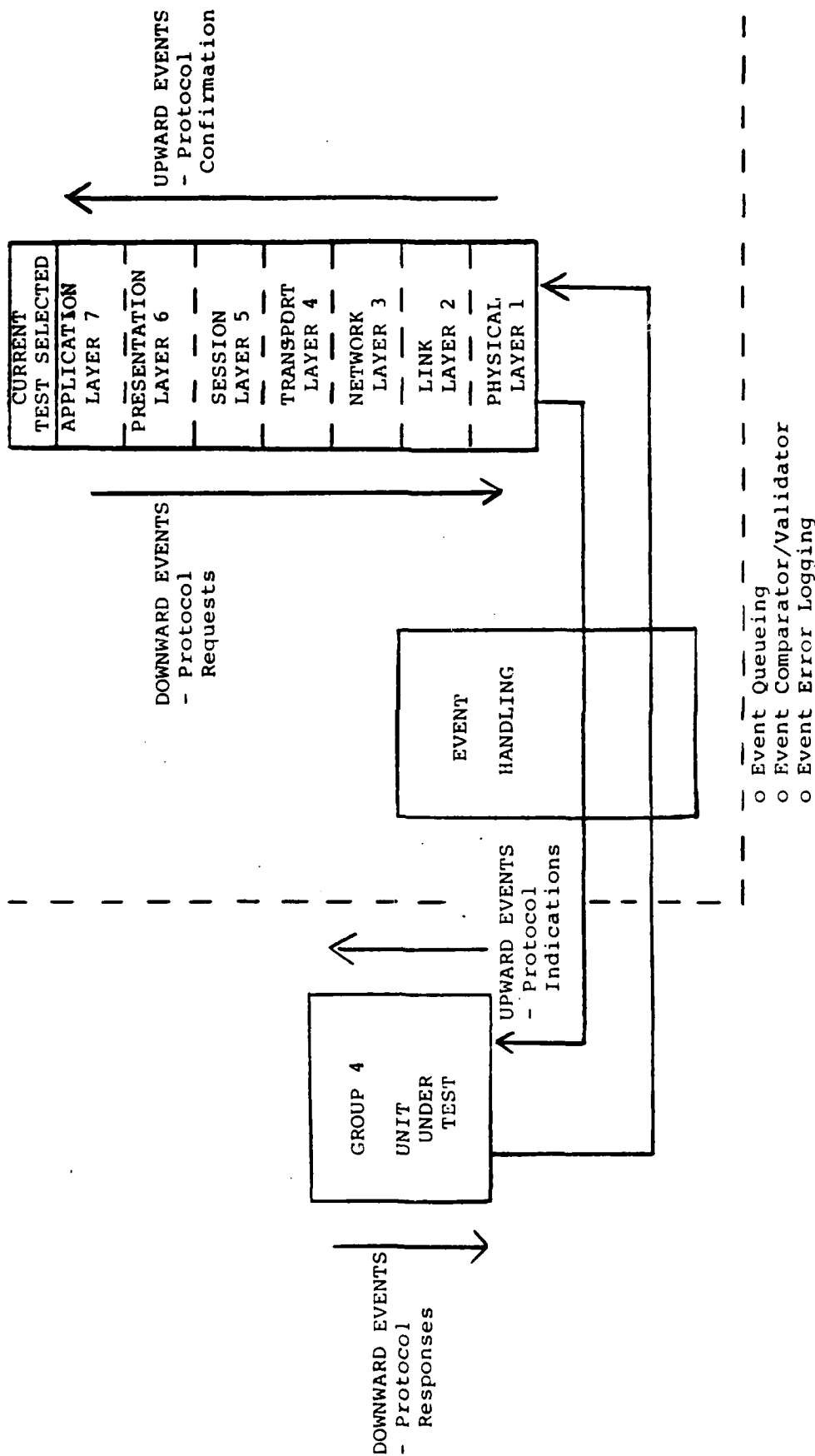


FIGURE 2-1

GROUP 4 VALIDATION SYSTEM

GROUP 4 TERMINAL/UUT

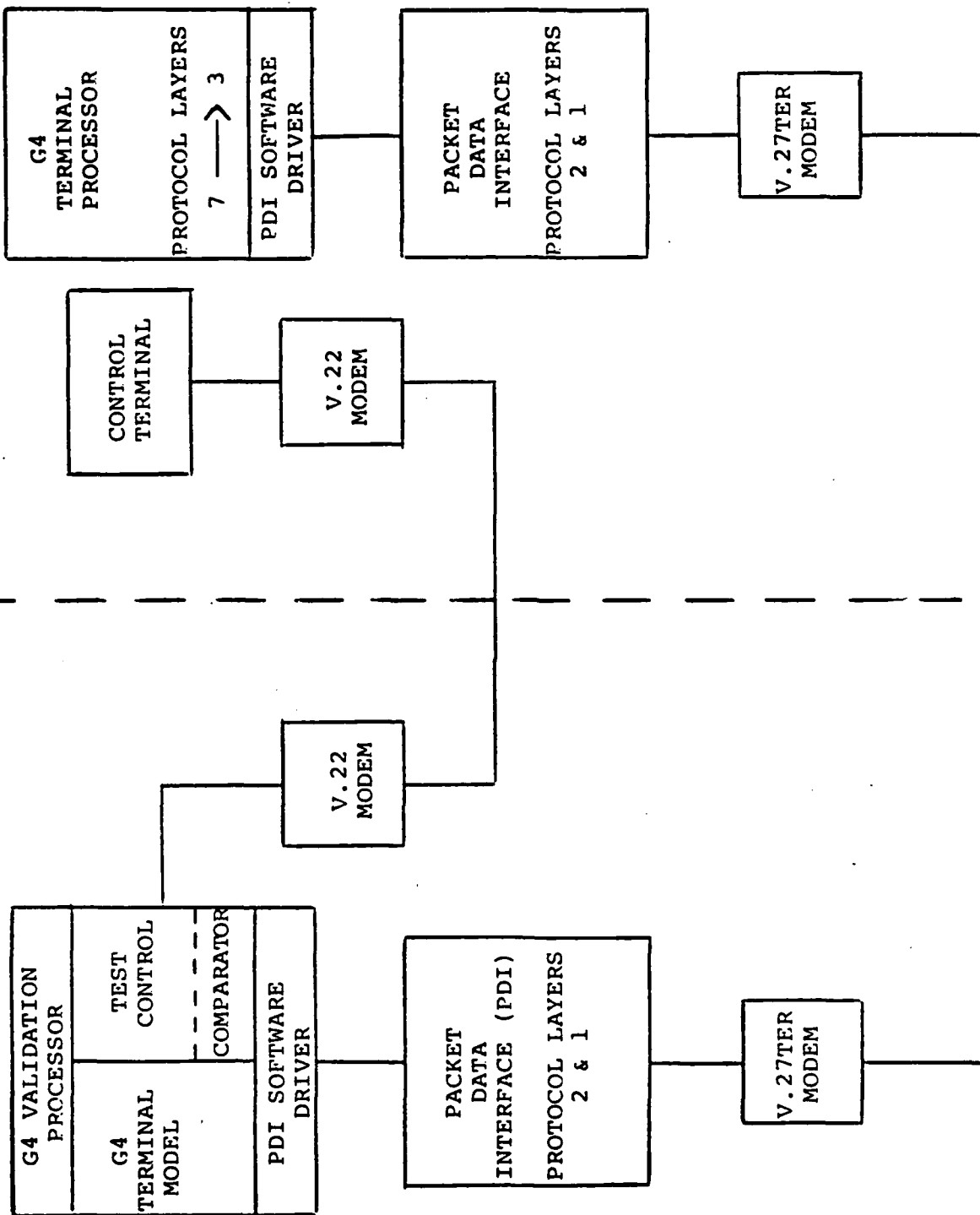


FIGURE 2-2

Group 4 Unit Under Test (UUT) side(s). The system executive controls the execution of the selected test(s) and verifies that the Group 4 UUT is correctly executing and responding to the test. The emulator simulates a "model" Group 4 terminal and performs all operations as requested by the validator. A system comparator compares the results of the model and the UUT insuring correct operation. The validation system software as implemented can support both the validator and emulator/UUT sides within the same processor for testing or two separate processors to demonstrate the final system operation. The software for the validation system was written in Fortran 77 to maximize its transportability between the HP 1000 and the DEC PDP 11/60. The software design and development was done using top-down structured techniques. This design approach yields software which is highly modular and easy to modify. This modularity will also facilitate changes to the software required by the evolving CCITT Recommendations.

The Group 4 Validation System hardware consists of the Packet Data Interface board, the V.27TER modem and the PSTN/PSPDN network connection. In addition the processor which is functioning as the validator has a 1200 bps dial-in control port in order to allow the user to communicate with the validation test set. The Packet Data Interface (PDI) board is a DIS-developed assembly designed specifically to support both the Group 4 validator and emulator functions. The design uses identical hardware to interface to both processors (currently a HP1000 and DEC 11/60). Module operation is controlled by an imbedded microprocessor executing code developed in a high level language ("C") which permits modularity and flexibility.

The PDI interfaces to the computer by parallel program controlled transfer interfaces. In the case of the HP 1000, this interface was also developed by DIS to permit the same interconnect to be presented to the PDI as that seen from the DEC 11/60.

### 3.0 Validation System Design

#### 3.1 Software Design Approach

##### 3.1.1 System Concept

Since the basis for Group 4 Facsimile equipment is the Teletex protocols as described in the CCITT recommendations listed in Table 3.1, the validation system software's primary responsibility is the implementation and validation of these protocols. Along with this requirement, the validation system must also be capable of specifying and testing the different parameters/variables allowed within each of the protocol layers as defined in the recommendation for that layer.

Shown in Figure 3-1 is a functional block diagram of the validation system software. From an overall point of view, the validation system drives two (2) operations - the UUT and the G4 terminal emulator - and compares the results. The emulator acts as a "golden" model against which the performance of the UUT is compared, giving due allowance for permissible variations in operation. By substituting another instantiation of the emulation and its interface, the validation software itself can be tested, with the help of both proper and selected improper variation controls applied to the "UUT" emulation.

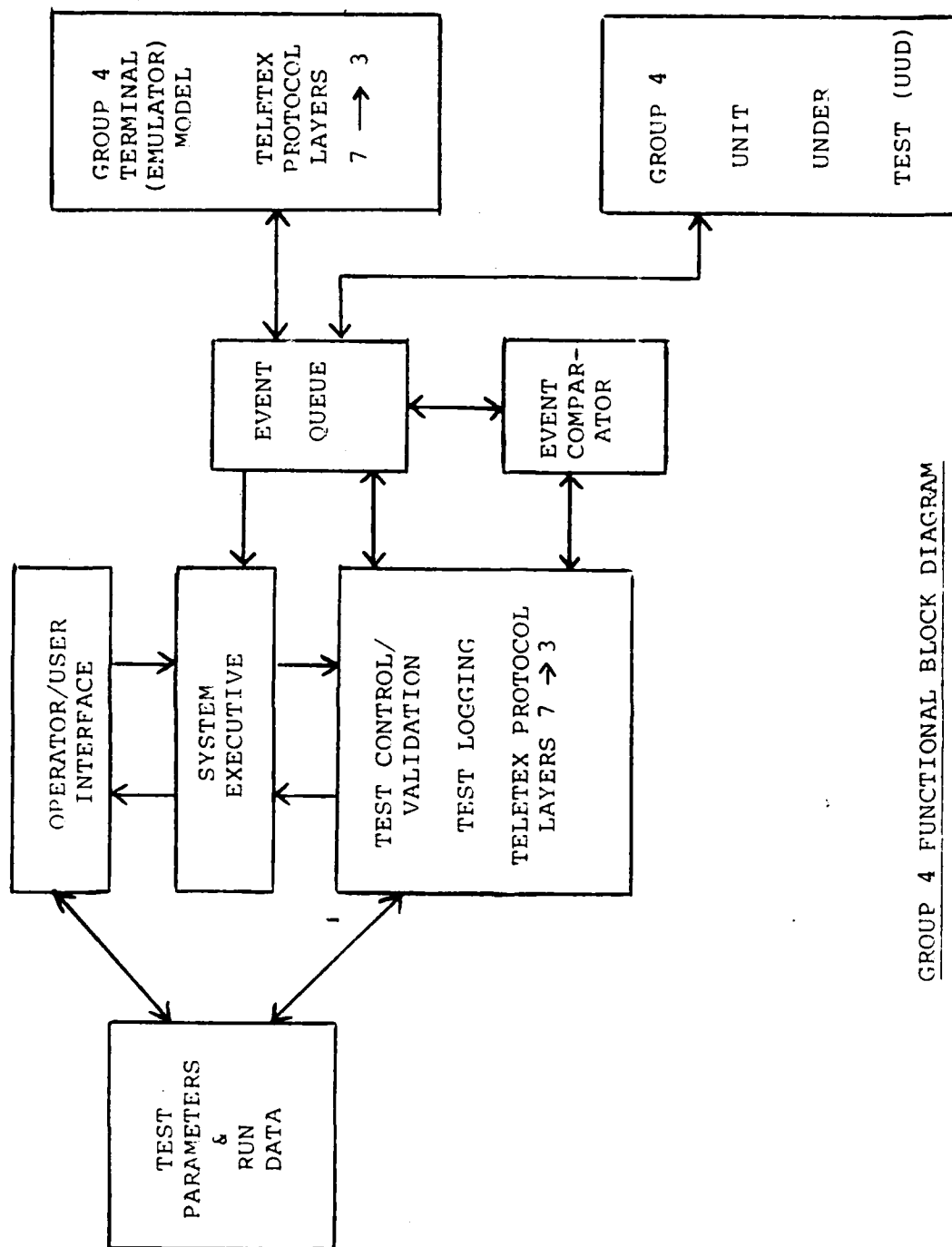
In operation, the system executive functions as the user layer (a psuedo layer 7.5), along with the user interface and the test package data. An event queue functions as the command and data channel, in both directions, between the system executive and the two validation instantiations; the queue, in effect, functions as the link between

TABLE 3.1

CCITT RECOMMENDATIONS BY ISO LAYER  
FOR GROUP 4 FACSIMILE TELETEX

APPLICATION LAYER NO CURRENT RECOMMENDATION		LAYER 7
PRESENTATION LAYER CCITT T.73		LAYER 6
SESSION LAYER CCITT T.62		LAYER 5
TRANSPORT LAYER CCITT T.70		LAYER 4
NETWORK LAYER CCITT X.25		LAYER 3
LINK LAYER CCITT X.25		LAYER 2
PHYSICAL X.24 PSTN	LAYER X.21 PSPDN	LAYER 1





GROUP 4 FUNCTIONAL BLOCK DIAGRAM

FIGURE 3-1

the software portion of the system resident on the validation processor and the UUT.

The system executive starts and stops processes, in response to commands via the user interface and completion (successful or not) indications from the validations, and also examines the event queue to determine which modules to poll for action. Each module, when polled, modifies a state (or substate) or moves data as appropriate; the comparator is then called to determine if the action taken was permissible, via comparison with the "good" model, in some cases forcing the latter to match the actual UUT's (or in test mode, the possibly faulted validation's) action, if a permissible variation.

#### 3.1.1.1 DESIGN PHILOSOPHY

Thoroughness of testing, and the fineness of detail in the results obtained from tests, were the driving principles in the design of the system. This consideration, reinforced by structured programming principles, dictated that all actions which are significant (in protocol terms) be made visible by small modules. In order to make the system capable of maintenance and enhancement, modules were aligned to the layering (in the OSI sense) and to the CCITT protocol recommendations which they implement. In this way, the scope of the system as a whole can be easily broadened, and modifications in the CCITT protocols can be incorporated with relative facility.

Fortran 77 was chosen as source language for the validation system because of its portability, efficiency, and its structuring

(especially with MIL-STD-1753 enhancements). Its support for serially reusable modules permits, and redundancy/inconsistency considerations strongly advise, modules dedicated to protocol functions which are usable by both the UUT and the emulation subsystems, on both sides of the interface. Fortran's COMMON blocks are used to store states, substates, and overall status. The event queue is used not only as a "mailbox" between modules for interlevel commands, but also as a journal for the actions taken by the UUT and emulations of it. With the use of multiple linked lists, each corresponding to a specific event, maintained by well tried "heap" storage management techniques, each module quickly performs its intended function.

In coding the various modules, strict adherence to ANSI X3.9-1978 extended by MIL-STD-1753, both in letter and spirit, was followed. This policy not only guarantees relative ease in porting software from processor to processor but also guarantees the reusability of modules as mentioned above. Module reentrancy was not relied upon, but has been taken advantage of (usually by multitasking) where available.

#### 3.1.1.2 DESIGN APPROACH

As noted in the proposal, a top-down methodology was followed in the software design of the system. The system itself was structured as "open-ended", using small modules. Each module when coded was essentially complete, but those of its functions which were not needed in the present description of the system resulted in direct or indirect invocations of "stubs" or placeholder modules. When it is necessary to add a given function to the system, the modules completing that function can be substituted for the stubs.

The event queue approach was chosen to bring as many protocol actions as feasible out into the open, where the performance of the UUT can be compared in detail with a properly acting model supposedly compatible with it. It permits detailed error reports which not only back up invalidity decisions but also aid the agency requesting the validation, without the use of difficult-to-implement protocol conformance evaluations in the large. The approach also enhanced the open-endedness of the system, particularly with regard to added functionality and CCITT recommendation revisions (e.g. S.62/T.62,T.73).

Other service routines were specified and implemented as the need for them became apparent.

### 3.1.2 VALIDATION SYSTEM DESCRIPTION

From a functional point of view, the validation system is comprised of the following parts:

- o Operator (user) interface
- o Test Package
- o System executive
- o UUT subsystem and status/buffer stores
- o Emulator subsystem and status/buffer stores
- o UUT/Emulator comparator
- o Event queue and allied management software

The following details the part played in the operation by each of these subsystems.

#### 3.1.2.1 OPERATOR INTERFACE

This set of modules provides the means by which specific tests can be selected and initiated, and the results of tests returned, in the form and in detail requested by the operator. It also provides the operator step-by-step instructions for normal UUT validations, including selection of alternate protocols where appropriate. For maintenance and diagnostic purposes, the operator may also choose between parallel and serial UUT and emulation operation, and may also compare a selectively faulted emulation with an unfaulted model. Internal to the system, this subsystem maintains a table of testing options, and calls the system executive to start, resume, or wrap up a test as instructed.

#### 3.1.2.2 TEST PACKAGE

While no modules are strictly part of the test package, this must be considered part of the system as a whole. This package will normally reside on auxiliary storage, and can be easily substituted for special purposes. Basically, it consists of test data for transmission, plus control information for selecting modes of operation on various levels. These modes include not only permissible alternatives but also invalid ones to test the UUT's capability to react properly to protocol errors.

#### 3.1.2.3 SYSTEM EXECUTIVE

When invoked by the operator interface, the system executive initiates the selected task by obtaining testing information from the

test package, storing it and initializing the event queue as appropriate. Thereafter, it polls the linked lists which make up the event queue and calls the appropriate modules to take action. On test completion or an abort condition signaled by the comparator subsystem, the general nature of the test result is passed to the operator interface, so that detailed test result data can be printed or otherwise provided.

Executive polling can be done in two ways, roughly describable as parallel and serial. In the parallel mode, the UUT and the emulation are kept essentially in step with one another; the UUT is blocked from getting more than a single significant protocol event ahead of the emulation. In this mode, the UUT does not proceed to step N+1 until the emulator has taken step N and its action compared with the corresponding UUT step. This mode is particularly useful for detailed examination of operation, especially in debugging.

The serial mode, on the other hand, allows UUT actions to have priority over emulation actions, letting the UUT "run free", so to speak. The emulator, and the comparison of its actions with the UUT, are handled as time is available, using the event queue as the UUT "history" medium. Serial mode may be required for some terminals and modes of operation; for emulation-to-emulation comparisons, the parallel mode is obviously preferable.

#### 3.1.2.4 UUT SUBSYSTEM

This subsystem consists of a set of modules, shared with the emulator subsystem, which implement the protocols and functions called

for at each layer of the OSI model. What actually dedicates it to the UUT (or an emulation of one) is the functional incorporation of the UUT and its hardware interface into the system, and the stored status, buffers, and linked lists specifically associated to the UUT or its standin. The procedure modules, as such, function as routines dedicated to their protocol implementation and transmission functions, rather than that of interfacing to the UUT or providing a control against which the UUT performance can be evaluated.

Each module performs one or more actions corresponding either to protocol-specified change of state or substate, or performs inter-layer translation functions. A module is thereby identified with specific sections or subsections of the protocol recommendation which it implements; its actions, rather than being "hard-coded", are driven by a decision table closely reflecting the "state diagrams" (when available) included with the CCITT recommendations, even to the state numbers and other annotations. The decision tables, fixed at compile time, are supplemented by parallel mask and vector tables which can modify actions either to reflect alternative actions or transmission paths, or to force improper actions to be taken, either to test the UUT's reaction to them, or for debugging purposes, especially in emulation vs emulation tests. These supplementary tables are modified during execution to implement alternate protocol choices, hardware vs software module implementations, and "error-force" option selections.

The heart of the UUT subsystem, as such, is the stored data which reflects the current states and substates of the UUT transmission in progress, the linked lists containing, by layer and direction, the

history of the transmission, and the transmitted data itself.

Substates, as far as the software is concerned, simply provide a more detailed description than the CCITT-defined states as such; the designation was chosen to keep the coarser states in line with the CCITT specifications. Relative to states, substates record such details as timeout counts, intermediate status in combine/divide operations, and other data needed to define fully the status of a given transmission.

The linked lists provide the main mechanism by which the protocol modules communicate with one another. In operation, the executive polls all linked lists for unhandled events; when one is found, the module "handles" the event, usually marks it as "done" and places another event on another linked list for some other module to handle, modifying the stored status accordingly. In cases where the correspondence between "input" and "output" events is not one-to-one, the module may delay "signing off" on the input event or placing another event on its own input linked list as is appropriate, to guarantee that it be polled to complete its function. In any case, the action taken by a protocol module on one invocation is scaled down to a maximum of one event in or out.

#### 3.1.2.5 EMULATOR SUBSYSTEM

The emulator subsystem shares all its protocol procedure modules (except where replaced by hardware/firmware links) with the UUT subsystem; the difference is in the status and buffer stores dedicated to the subsystem, the linked lists which provide layer interfaces, and the method by which supervisory control over it is exercised. Each



protocol module is ignorant whether it is performing its function for the UUT or emulator, but is provided with the status tables, linked lists, etc., peculiar to one or the other by the calling executive. The basic decision tables used are the same as for the UUT, but supplementary mask and vector tables may differ, according to the purpose of particular tests, and the double use of emulator modules on both the DTE and DCE sides of the transmission, for example when the effect of one sided protocol errors are being tested. The difference in supervisory control is implemented by the comparator subsystem, working with the executive, as described below.

#### 3.1.2.6 UUT/EMULATOR COMPARATOR

The job of the comparator subsystem is to keep the UUT and emulator systems in line with another, comparing their actions on an event by event basis, and reporting on serious discrepancies. In order to do this, actions taken by protocol modules on both sides are "filtered" through a comparator module which takes differential actions, depending on the "side" from which the action emanates. On the UUT side, in serial mode actions are allowed to proceed; in parallel mode, an action may be held up until the emulator side has reacted to the corresponding stimulus. This is accomplished by the protocol module placing its generated events on the comparator module's stimulus list; the comparator will pass it on (by relinking) when and if appropriate.

Once both sides have reacted to corresponding stimuli, results are compared. If they match, the process is allowed to continue with no special action being taken. If the action taken by the UUT side is

a valid alternative to that of the emulator, the latter's action is modified to match the UUT. Otherwise, an error report is generated and the entire process is aborted or modified and forced to continue as is appropriate to the seriousness of the error and pertinent operational modes as set by the operator.

In cases where only one side is a software module whose stimuli and responses are available to the comparator (for example, the DTE-end emulator paralleled to the UUT itself), no attempt is made to keep actions in line. Instead, the "small" actions on the "soft" side are assumed to match the other side, and alignment and comparison are deferred to a layer where both sides are available. No error reports are generated where mismatches can not be diagnosed, of course; however, the comparator subsystem, through suitable interface modules, will merge error detections passed on by hardware/firmware modules into the same error report list used by software/software mismatch reports.

#### 3.1.2.7 EVENT QUEUE AND ASSOCIATED MANAGEMENT FACILITIES

Although each event "belongs" to a specific side, layer, and module at any given time, its structure is common to the overall process itself. Storage for each event is allocated in space available to all modules, via common allocation/garbage collection routines. For auditing purposes, all events are linked by time of generation; for functional purposes, a second linkage is used to connect them with previous and successive events at the same and neighboring layers. This latter connection is modified (by relinking) by comparator modules in order to let a process continue or to "roll

back" a given action.

Each event contains, at a minimum, the following information:

- (1) An indication of the nature of the event, including codes for layers involved;
- (2) An indication of the "side" (UUT vs. emulation), "end" (DCE vs. DTE), and direction of the event;
- (3) Clock time for the event;
- (4) List linkages (on time and functional bases);
- (5) Linkages to data, where appropriate;
- (6) Indications of state changes associated with the event.

Associated with the event queue are not only the allocation routines mentioned above, but also other service routines performing such chores as relinking lists and similar functions. A real time clock of adequate precision is used to provide event timing for timeout sequences and similar functions. Similarly, for reporting purposes, routines are required for editing compact error reports from the comparator subsystem into terms the operator can recognize.

## 3.2 Hardware Design Approach

### 3.2.1 Overview

The physical and link layer that implement HDLC/LAPB were most conveniently implemented with a microprocessor and appropriate protocol chips. Specifically, the 68000 16 bit processor was chosen for processor power, data bus size, convenience of vectored interrupts and high level language development support. The WD2511 protocol chip (manufactured by Western Digital) presented the best match of capabilities for the 68000. It contains three internal microcontrollers.

The basic design approach is to use the 68000 processor to handle packet transmission between the host computer and an on-board memory. The WD2511 in turn moves all data between memory and the modem in the bit-oriented, full duplex serial format which conforms to CCITT X.25 with programmable enhancements. The 68000 processor also controls setup of the WD2511 and modem operations, such as dialing and half duplex operations as appropriate to operate on either PSTN or PSPDN.

### 3.2.2 PDI Characteristics

The PDI board is configured as follows:

- MC68000 Processor, 16 bit, 6Mhz
- Program storage: 16K words of EPROM
- RAM memory: 128K bytes of dynamic RAM
- 16 bit bi-directional parallel data  
interface with host

- Two asynchronous, 9600 baud, serial RS232 ports for test purposes
- WD2511 X.25 packet network interface, providing DMA transfers of data
- 1 millisecond timer to facilitate half duplex operations per T.71
- Modem controls for data handling and dialing
- Constructed on DEC baseboard, Hex size

### 3.2.3 DEC 11/60 Parallel Data Interface

Configuration, send, receive, and status data is transferred between the PDI and the DEC computer using a type DR11-C general purpose interface card. This is installed internally to the PDP 11/60 and provide a 16 bit data path. Data is moved under program control only, this is not a DMA type interface. All data transfers are made in a block format which includes block identifier, length, and checksum to insure the integrity of the transfers. A single 50 conductor cable interconnects the DR11-C and the PDI, which is also mounted within the PDP 11/60 card frame. In this configuration, the PDI only draws power from the UnibusR/.

### 3.2.4 HP 1000 Parallel Data Interface

When the PDI is connected to an HP 1000 computer, it resides in a 3 1/2 inch high, rack mounted chassis and interconnects with a

---

R/Unibus is a registered trademark of Digital Equipment Corporation.

DIS-developed HP interface. The connection is made by a 50 conductor ribbon cable. The HP interface card mounts in the I/O backplane and provides identical functions as the DR11-C. Interrupt-driven I/O channel driver software moves command, status, and data blocks between the level 3 Fortran program and the PDI module.

#### 4.0 Validation System Operation

The performance of a test by the validator can be roughly divided into five phases:

- Test selection
- Test validation
- Test configuration
- Test execution
- Test post-mortems

The following paragraphs note the actions taken in each of these phases and the specific inputs and parameters required for each.

##### 4.1 Test Selection

This phase involves the determination of the specific test to be performed and any "special" parameters involved. It involves the processing of a "test select" file which names the test to be used, and which supplies the parameters unique to this particular performance. The phase is satisfactorily completed when the test name has been determined and the other test select parameters processed without serious errors.

The test select phase opens with a request to the operator to supply the name of the "test select" file. If the test involved is "on file", the operator supplies the file name and other necessary directory information at this time; the test select information in this file is then processed. The operator may alternately enter a

carriage return only, which causes the validator to expect the test select information "on line" from the operator. In this case, the operator types in test select parameters, terminating them with a dollar sign (\$) when done. See Appendix C for a description of test selection syntax.

The purpose of the test select file (as opposed to the test file itself) is to identify the test [file] involved and specify parameters which supplement or override those in the parameter portion of the test file. In other words, the test select parameters need only be the "unusual" or "different" values needed for a particular performance.

The test name may be specified in any one of the following forms (see Appendix C for detailed syntax) in the test select file:

TEST='filename,etc...'

TEST=\*

TEST=&

The first form carries the file and/or unit identifier for the test file to be used. It may also carry a "version" identifier to distinguish between multiple tests stacked in a single file. This form (except as noted below) indicates that the named test is to be read from the file specified and its parameters are to be used as the "base" to which the test select file parameters are to be applied.

The second and third forms specify that no test file "read" is necessary, and may therefore only be used immediately following



successful configuration and execution of the specific test to be used. Both use as "base" the previously executed test. The asterisk (TEST=\*) form specifies that the test is to be repeated exactly as last configured, except for the variations specified in the test select file; the ampersand (TEST=&) form, on the other hand, indicates that the "base" to which the test select parameters are to be applied is the test as originally read in, i.e., the test restored to its "default" condition.

For logging purposes, the "repeat" and "default" options may also be specified using the first (explicit) form above, by following the "TEST=" parameter by a "REPEAT" or "DEFAULT" parameter respectively. For example, assuming that the test just executed was identified as "TEST1.ABC:3",

```
TEST='TEST1.ABC:3',DEFAULT
```

is equivalent to

```
TEST=&
```

for test selection purposes. A "NEW" parameter is also recognized, to permit explicit indication that the test must be read from the file (in the validation phase) before configuration.

Shown on the following pages are examples of both test selection and test files. On page 4-5 are listings of two different test selection files. Each test selection file consists of three separate tests identified by a version number. The first two tests modify a

single variable in the test file G4BASIC.TST. The third test repeats the previously executed test with an additional variable change. On pages 4-6&7 is a listing of the test file G4BASIC.TST. This test file is comprised of three parts;

- side, end, and layer configuration
- layer specific variable establishment
- test run data

This test file in conjunction with the changes specified in the appropriate test selection file will completely define the validation test to be executed.

#### 4.2 Test Validation

Test validation involves verifying that the test file named, if "new", is in the proper form, and processing its (default) parameters for later merging with those specified by the test select file. If the "repeat" (TEST=\*) or "default" (TEST=&) options are chosen, no validation is necessary except confirming the existence of a previously configured test. The success or failure of the validation phase is indicated by an appropriate printout.

#### 4.3 Test Configuration

This is the most complex of the three "test setup" phases. First, the "repeat" (\*) and "default" (&) parameter "values" specified in the test select file are "filled in" from the last-executed test performance or the "as read in" test parameters as appropriate. The test select parameters are then merged with either those for the last performance (if test repeat is specified) or those of the "standard"

### TEST SELECTION FILE EXAMPLES

Name: G4LYR5MOD.SEL

Function: To execute the Group 4 Validation System Base test with modifications to the Session Layer (5) Inactivity timer (CT1T) and Window Size (CWSZ)

#### File Contents

```
$TEST_SELECTION V=2.04.01
TEST=G4BASIC.TST V=1.00.00           ! G4 VALIDATION BASE
SIDE=(EMUL, END=(BOTH, LAYER=(5, CT1T=45))) ! change timer
SIDE=(UUT, END=(TEST, LAYER=(5, CT1T=45))) ! value
$
$TEST_SELECTION V=2.04.02
TEST=G4BASIC.TST V=1.00.00           ! G4 VALIDATION BASE
SIDE=(EMUL, END=(BOTH, LAYER=(5, CWSZ=2))) ! change window
SIDE=(UUT, END=(TEST, LAYER=(5, CWSZ=2))) ! size value
$
$TEST_SELECTION V=2.04.03
TEST=*                                ! G4 BASE WITH CWSZ=2
SIDE=(EMUL, END=(BOTH, LAYER=(5, CT1T=45))) ! change timer value
SIDE=(UUT, END=(TEST, LAYER=(5, CT1T=45))) ! using prev test
$
```

Name: G4LYR3MOD.SEL

Function: To execute the Group 4 Validation System Base test with modifications to the Network Layer (3) Data Packet Size (CDPS) and Use Delivery Confirm (QUDC).

#### File Contents

```
$TEST_SELECTION V=2.04.01
TEST=G4BASIC.TST V=1.00.00           ! G4 VALIDATION BASE
SIDE=(EMUL, END=(BOTH, LAYER=(3, CDPS=256))) ! change data
SIDE=(UUT, END=(TEST, LAYER=(3, CT1T=256))) ! packet size
$
$TEST_SELECTION V=2.04.02
TEST=G4BASIC.TST V=1.00.00           ! G4 VALIDATION BASE
SIDE=(EMUL, END=(BOTH, LAYER=(3, QUDC=Y))) ! change to use
SIDE=(UUT, END=(TEST, LAYER=(3, QUDC=Y))) ! delivery confirm
$
$TEST_SELECTION V=2.04.03
TEST=*                                ! G4 BASE WITH QUDC=Y
SIDE=(EMUL, END=(BOTH, LAYER=(3, CDPS=256))) ! change packet size
SIDE=(UUT, END=(TEST, LAYER=(3, CDPS=256))) ! using prev test
$
```

# TEST FILE EXAMPLE

Name: G4BASIC.TST

Function: To define the side, end and layer configuration for the Group 4 Validation System Base Test. It also defines the default values for the layer specific variables as established in the CCITT T & X Recommendations for Group 4 Facsimile.

## File Contents

### \$TEST\_PARAMETERS

```
!
! establish side, end and layer configuration
!
SIDE= (EMUL, END= (BOTH, LAYER= (U, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (7, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (G, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (6, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (F, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (5, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (E, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (4, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (D, CNFIG=S)))
SIDE= (EMUL, END= (BOTH, LAYER= (3, CNFIG=S)))

SIDE= (UUT, END= (NEAR, LAYER= (U, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (7, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (G, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (6, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (F, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (5, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (E, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (4, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (D, CNFIG=S)))
SIDE= (UUT, END= (NEAR, LAYER= (3, CNFIG=S)))

!
! establish layer specific variables
! using T & X recommendation defaults
!
SIDE= (UUT, END= (NEAR, LAYER= (6, HBTC=2, HICF=0, HPGDW,10200
HPGDL=13200, HPTD=200)))
SIDE= (EMUL, END= (BOTH, LAYER= (6, HBTC=2, HICF=0, HPGDW,10200
HPGDL=13200, HPTD=200)))
SIDE= (UUT, END= (NEAR, LAYER= (5, CT1T=60, CT2T=60, CT3T=4,
QCTU=N, QAER=N, QUYO=N,
QULL=N, QRAJ=N, CMSZ=3, CWSZ=3,
PBLFCTD=256, PBLFCTU=256)))
SIDE= (EMUL, END= (BOTH, LAYER= (5, CT1T=60, CT2T=60, CT3T=4,
QTCU=N, QAER=N, QUYO=N,
QULL=N, QRAJ=N, CMWZ=3, CWSZ=3,
PBLFCTD=256, PBLFCTU=256)))
```

```

SIDE= (UUT, END= (NEAR, LAYER= (4, C02A=45, C03B=6, C11C=45,
                                C11R=5, C11T=225, CBRM=2,
                                CBAM=2, CUBS=7, CMBE=7,
                                PBLFCTD=128, PBLFCTU=128,
                                QANC=Y, QATC=Y, QTXP=Y,
                                QTRY=Y, CTCC=0, CREF='LOCREF'))))
SIDE= (EMUL, END= (BOTH, LAYER= (4, C02A=45, C03B=6, C11C=45,
                                C11R=5, C11T=225, CBRM=2,
                                CBAM=2, CUBS=7, CMBE=7,
                                PBLFCTD=128, PBLFCTU=128,
                                QANC=Y, QATC=Y, QTXP=Y,
                                QTRY=Y, CTCC=0, CREF='LOCREF'))))
SIDE= (UUT, END= (NEAR, LAYER= (3, CDPS=128, QXPS=N, QUDC=N,
                                CHNGRPD=0, CHNGRPU=0,
                                CHNNUMD=0, CHNNUMU=0)))
SIDE= (EMUL, END= (BOTH, LAYER= (3, CDPS=128, QXPS=N, QUDC=N,
                                CHNGRPD=0, CHNGRPU=0,
                                CHNNUMD=0, CHNNUMU=0)))

```

# \$TEST\_RUN\_DATA

```

!
! actual test run data
!

```

```

/QUEUE      E7AACQ  L=7UN  HPND      S=0<0/0
/QUEUE      E7AACQ  L=7CN  HPND      S=0<0/0

```

```

          ↓
/EVENT      E7AAIQ  L=7UN  HPND      D=-----
/EVENT      E7AAIQ  L=7CN  HPND      D=-----

```

```

          ↓
/END      PRINT  PURGE

```

(as read in) test. Specific test parameters are then processed, first the "global" ones which must be first processed, then those "local" to specific layers, sides, etc. The layer-specific "local" parameters must include configuration indications which specify how each layer and half-layer is implemented (e.g., accessible as software, inaccessible in hardware, an interface between the two, or just "not there"); the next stage in the configuration phase validates that the layer to layer configurations are consistent. Finally, any "leftover" parameters are processed and reported on.

Each stage in the configuration phase can detect and classify errors as "serious" (i.e., fatal) or "possible" (nonfatal). At the conclusion of the phase, summary counts of the two error types are reported. If no fatal errors have been detected, the operator is asked whether the test is to proceed to the execution phase. If the reply is affirmative, successful configuration of the test is reported, and the stage is set for test execution. However, if fatal errors have been detected, or the operator has responded negatively to the request to proceed, configuration failure is reported, and status is returned (as far as possible) to that prior to test selection.

Generally, errors are classified as "serious" if the test cannot be reasonably executed under the values (or lack of them) specified by the particular parameter causing the error. "Possible" errors, on the other hand, are usually those for which the module concerned can specify a reasonable "default" value (which it reports), or simply a (possibly misspelled or misplaced) parameter which it cannot recognize. The many handling modules themselves are responsible for the classification of the various errors involved.

There is, however, a class of serious errors which can be specified in the parameter portion of the test file itself. The test file can indicate that the test select file must not override certain parameter values, and also require the test select file to supply certain parameters. For specifics, see the FIXED, REQUIRED, and OPTIONAL parameters in Appendix A, but briefly it works like this:

```
...,REQUIRED=(.....),...
```

indicates that a serious error will be detected if the test select file fails to supply any parameters named within the parentheses;

```
...,FIXED=(.....),...
```

indicates that any attempt to override parameters named within the parentheses will be diagnosed as a fatal error; and

```
...,OPTIONAL=(.....),...
```

restores the options of supplying and/or overriding the parenthesized parameters as appropriate to the particular module or modules concerned.

#### 4.4 Test Execution

This phase involves initialization of global and layer-specific status to "startup" condition, preparation of the test "script" (run data) for processing, and finally transfer of control to event-driven test execution. Global initialization involves setting the event queue "empty" (ready to receive events), and resetting test-level status indicators. Initialization of the various layer/side/end

combinations depends on how each is configured.

For a layer/side/end combination configured via a protocol or translator module (accessible from both "upward" and "downward" directions), it is necessary only to reset its status to "startup" condition. For an interface combination, however, in which one of the vertical directions is "blocked" (i.e., inaccessible because implemented in hardware), it is necessary to install a "poll" event, which will invoke the module from time to time to check for transmissions coming from "outside" which would otherwise be ignored. For example, in a configuration where layers 3 and below are implemented by a layer 3 interface module which talks to an X.25 "chip", an "upward" poll event would be installed at layer 3 (to check for transmissions coming in via the chip), as well as a "downward" poll event at layer U (7.5) which causes the test run data (script) to be input to the system when appropriate.

Following state initialization, the designated test file is searched for its run data. If successful, test "script" read status is initialized; otherwise the test performance is aborted.

Assuming that all the "setup" steps have been successful, the actual execution cycle is entered. This involves searching the event queue for events yet to be processed, and processing them. The phase ends when no events remain to be processed, or a "quit" command is entered by the operator. Initially, the poll event at layer 7.5 causes the module implementing that layer to "read" the test "script" and enter appropriate events into the queue.



#### 4.5 Test Post-mortems

Following the successful or unsuccessful performance of a test, its results are reported. This involves printouts of both completed (fully processed) events, and events whose processing was not completed, either because of lack of time or error conditions raised. In addition, session statistics (primarily resource utilization) are reported. This phase also serves to prepare the system for the next test performance.

## APPENDIX A

### A.0 Validation System Software

Shown in Figure A-1 is a block diagram of the validation system software. The system software was divided into 3 major components: protocol processing routines, test selection and configuration routines and event processing routines. A fourth group of routines which are seen at the bottom center of Figure A-1 are the system service routines. These routines, as indicated by their names, support those functions within the system software that are required by the three major components listed above.

### A.1 Teletex Protocol Processing Routines

The protocol processing routines are those routines which implement the intra layer communication (e.g. layer 4 to layer 4 peer protocol) and the inter layer communication (e.g. layer 4 to layer 3 service primitives) as established for the Open Systems Interconnection (OSI) seven layer architecture (Figure A-2). Located between each of the protocol processing routine layers are translator routines (i.e. half layer routines) which perform all interlayer translation required.

Since the Teletex Protocol structure, as shown in Figure A-3, is based on the OSI architecture, the protocol processing routines are structured accordingly. The Teletex Protocols can be divided into two groups, one group containing the transport protocol (layers 1-4), and

GROUP 4 VALIDATION SYSTEM SOFTWARE

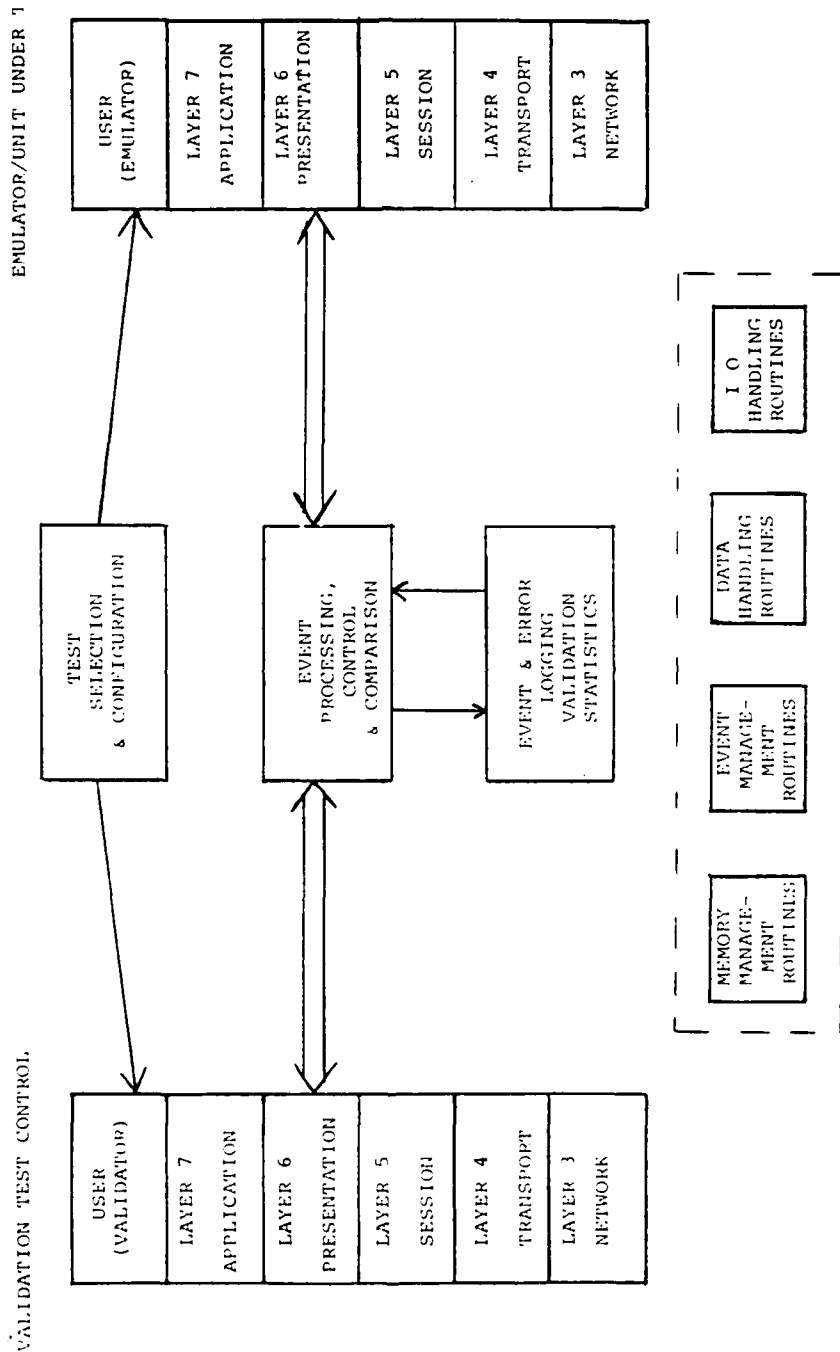
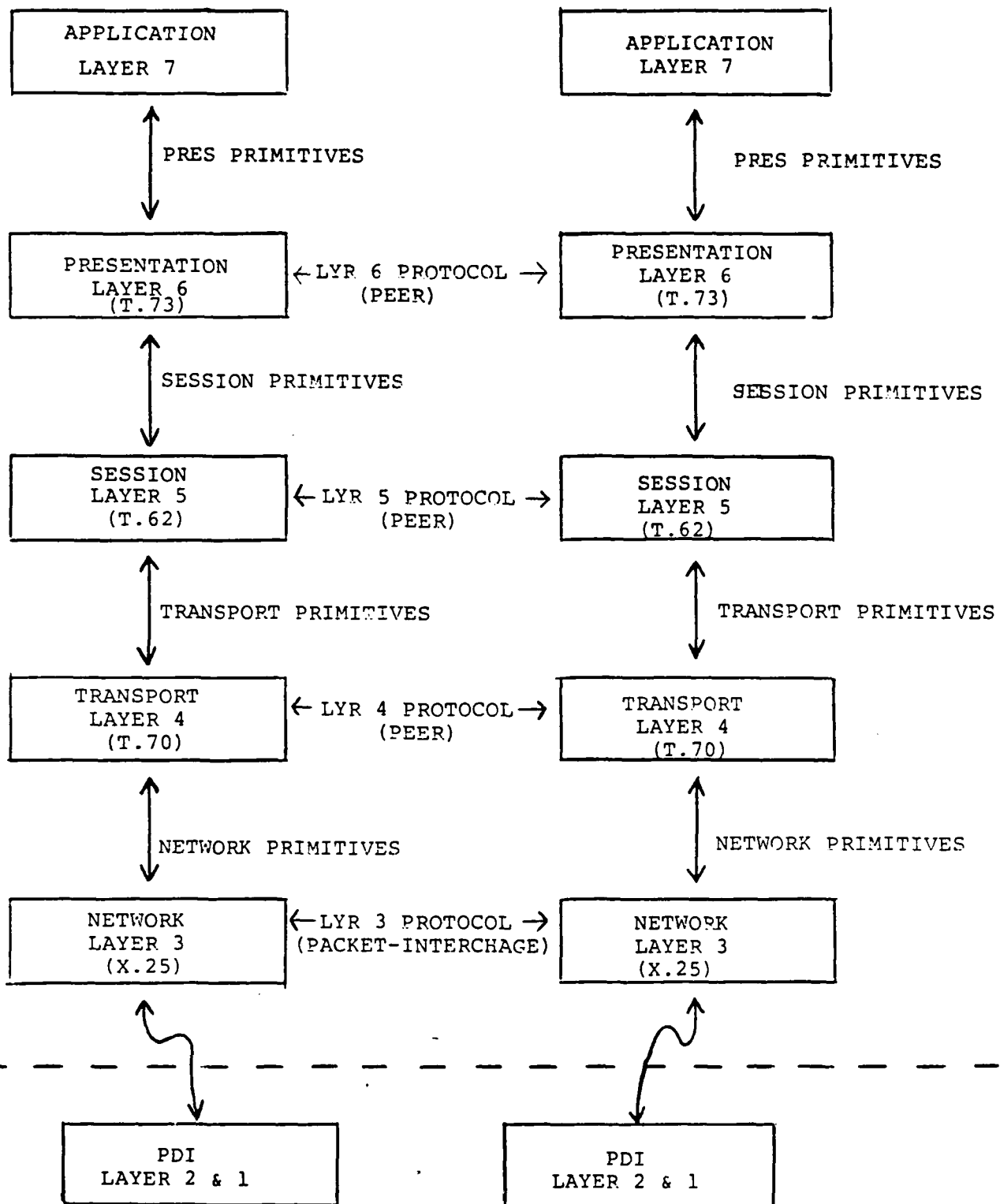


FIGURE A-1



PROTOCOL PROCESSING ROUTINE STRUCTURE

FIGURE A-2

A - 3

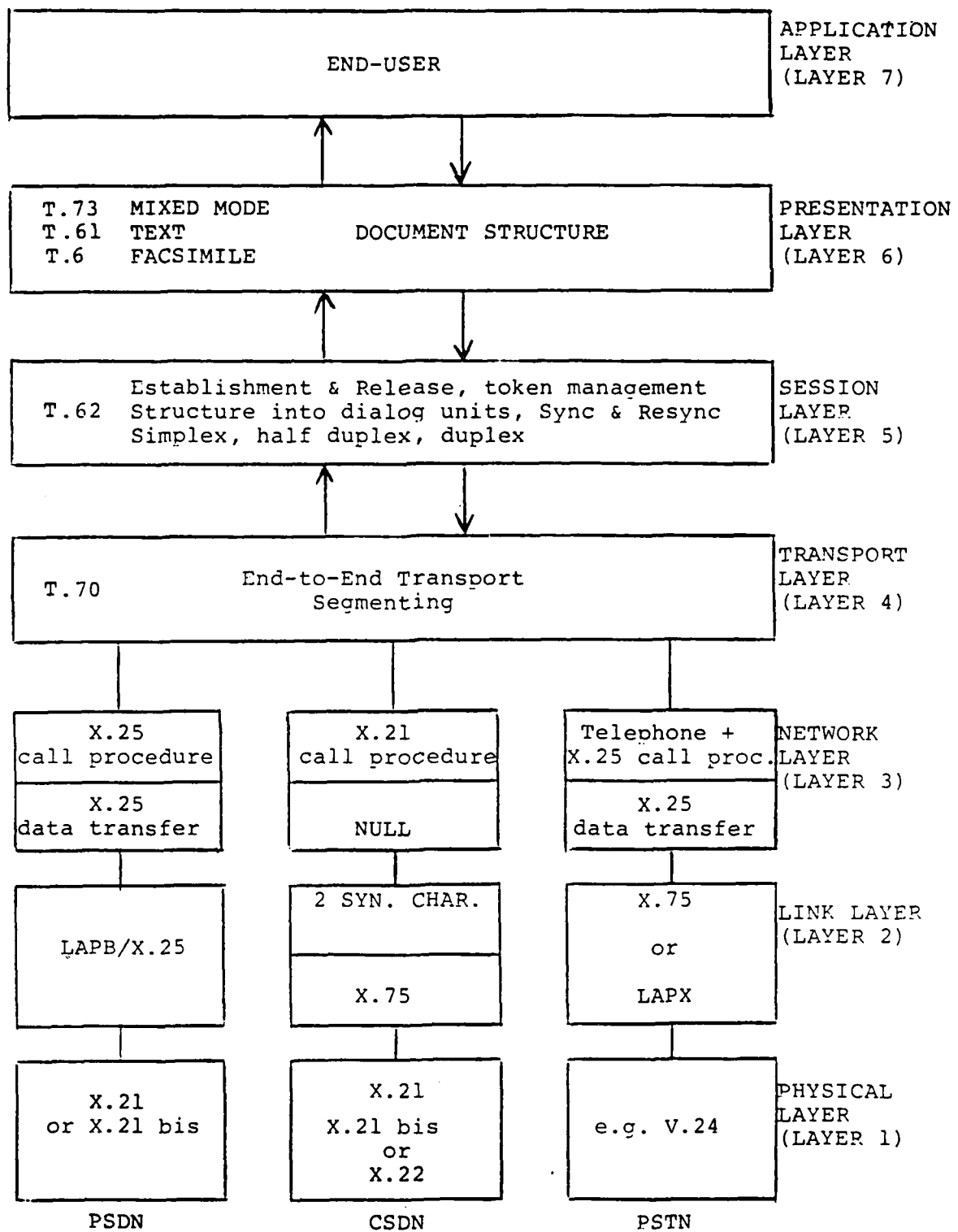


FIGURE A-3 TELETEX PROTOCOL STRUCTURE

the other group containing the user-oriented protocols (layers 5-7).

#### A.1.1 User-Oriented Protocols

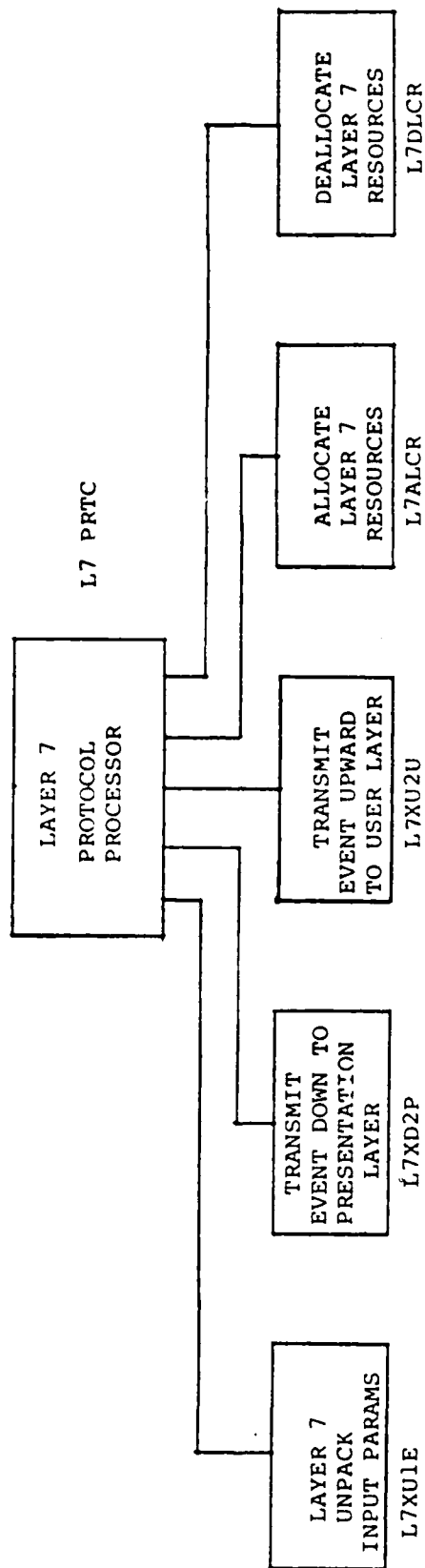
The user-oriented protocols are provided in layers five, six, and seven. These layers provide organized, synchronized, meaningful information exchange.

##### A.1.1.1 Application Layer - Layer 7

There is no current standard for the Application layer within the Teletex protocol. In order to maintain the seven layer structure a set of layer 7 routines was implemented as seen in the VTOC in Figure A-4. These layer 7 routines act primarily as a pass through from the user to the Presentation layer.

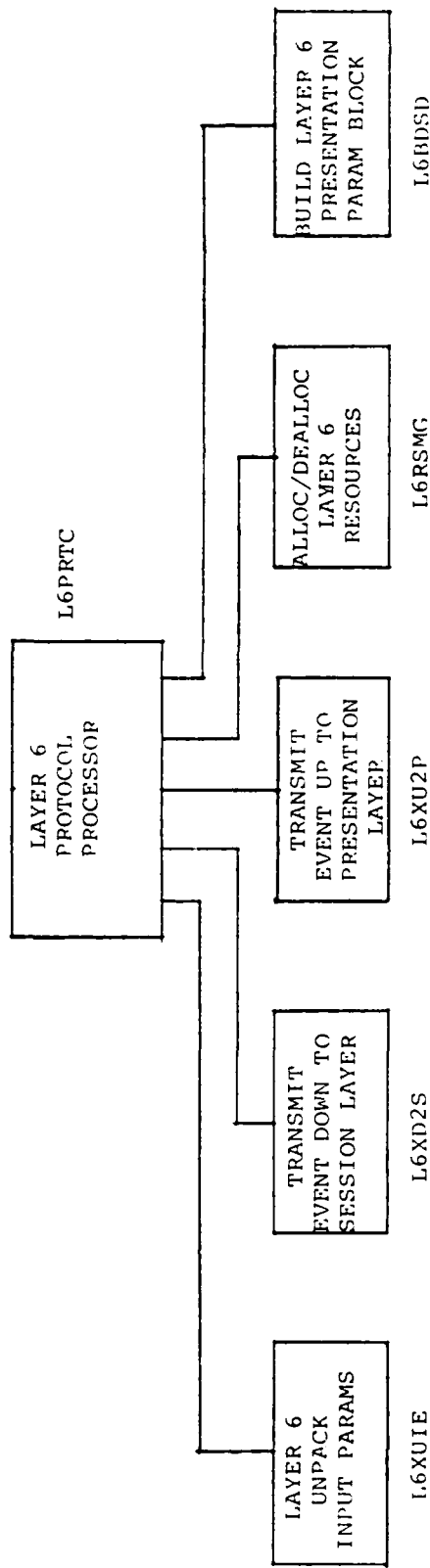
##### A.1.1.2 Presentation Layer - Layer 6

The function of the presentation layer is to define the characteristics of the structure of the information that users wish to exchange. At present there are three CCITT recommendations that address the Group 4 facsimile presentation layer: T.73 defines the document interchange protocol for Telematic services, T.6 defines the coding scheme for Group 4 Facsimile equipment, T.5 defines the Group 4 terminal equipment. A VTOC for layer 6 is shown in Figure A-5. All data conversion and compression/decompression is performed within layer 6 before transfer to the session layer.



LAYER 7 VTOC

FIGURE A-4



LAYER 6 VTOC

FIGURE A-5

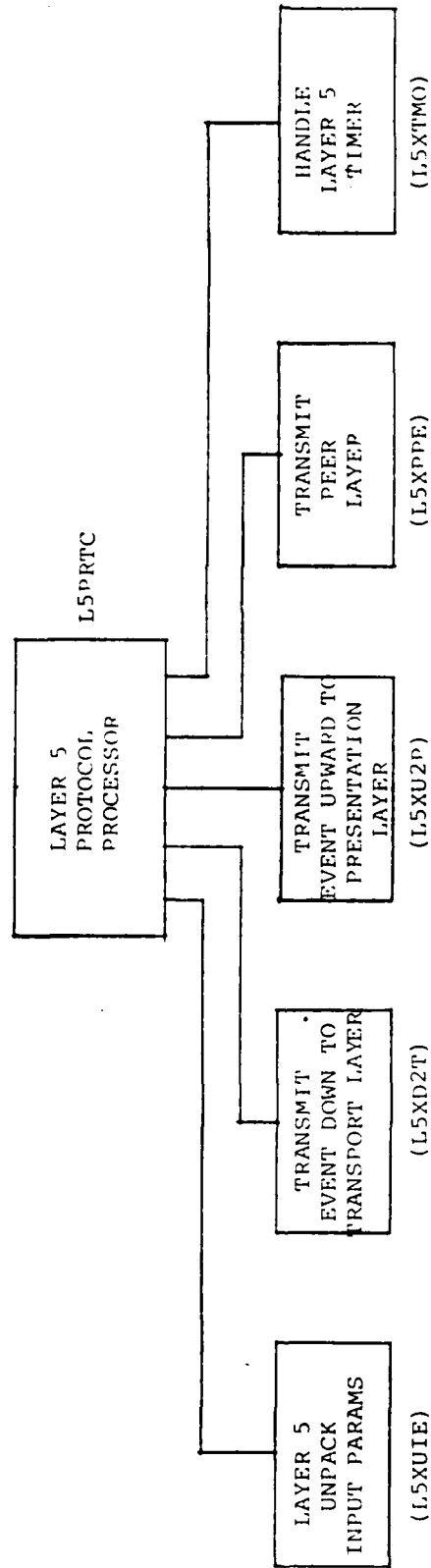


#### A.1.1.3 Session Layer - Layer 5

The session layer is responsible for setting up, managing and tearing down process-to-process connections including document synchronization and recovery. The protocol for this layer is defined by Recommendation T.62 which includes a "session" part and "document" part. It is convenient to treat both of these as an integral session layer protocol.

The session establishment function includes negotiation of parameters such as session profile, window size for checkpoints, and whether facsimile or mixed mode. Session establishment is followed by information exchange using the protocol element "Command Session User Information" (CSUI). Orderly session release is also initiated by command.

One of the main tasks of the session layer is to allow the structuring of the exchange of data into a series of dialog units which are isolated from one another by means of major synchronization points. The Teletex document can generally be interpreted as a dialog unit. Further synchronization marks can be inserted within a dialog unit to provide resumption points for a broken unit. These minor synchronization points are known as checkpoints. (End-of-page is an example.) The session layer also provides the mechanisms required for a reset in a defined state or a restart at a specific checkpoint. A detailed VTOC for layer 5 is shown in Figure A-5.



LAYER 5 VPROC

FIGURE A-6

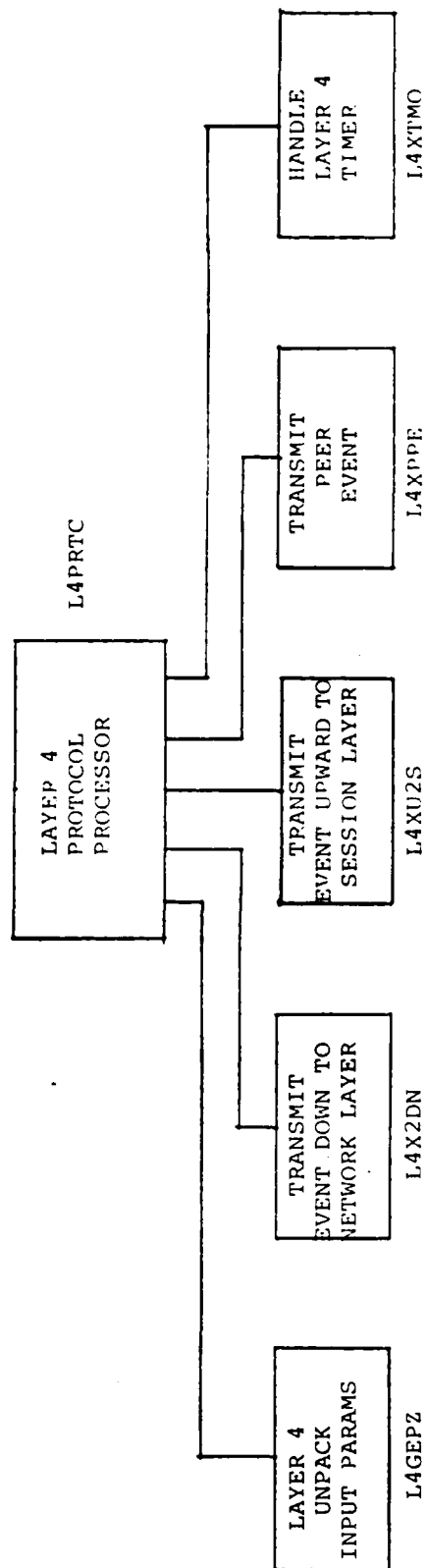
## A.1.2 Transport Protocols

### A.1.2.1 Transport Layer - Layer 4

The transport layer service is implemented according to CCITT Recommendation T.70. The object of the transport layer is a network-independent transparent transfer of data between systems. This layer serves as a bridge between the services performed by the network layer (3) and the services required by the session layer (5). It must hide all the details of the communication subnet from the session layer, so that for example, a point-to-point subnet can be replaced by a satellite link without affecting the session, presentation, or application layers. In addition to transport connection establishment, option negotiations and indication of procedural errors, the main function of layer 4 is segmentation. That is, in the data transfer phase the session layer passes on logically contiguous data blocks of unspecified length to the transport layer. The transport layer divides these blocks into blocks of predetermined length suitable for transmission while preserving their order. Shown in Figure A-7 is the VTOC for layer 4.

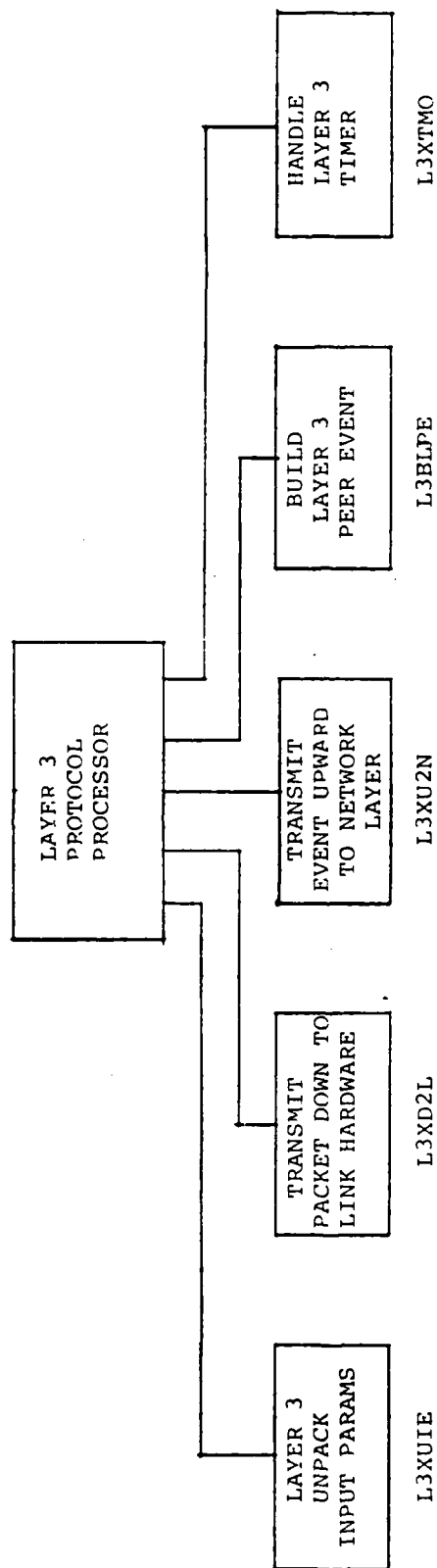
### A.1.2.2 Network Layer - Layer 3

The Network Layer service is implemented according to CCITT Recommendation X.25. The network layer provides the means to establish, maintain and terminate connections between systems containing communicating application processes. The basic layer service of the network layer is to provide the transparent transfer of all data submitted by the transport layer. The network layer is



LAYER 4 VTOC

FIGURE A-7



LAYER 3 VTOC

FIGURE A-8

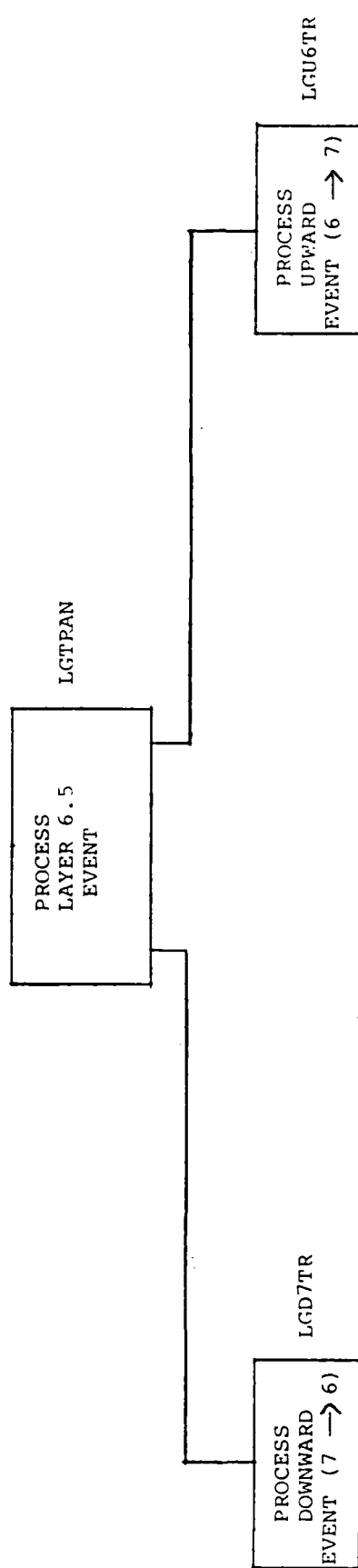
assumed to contain functions necessary to mask the differences in the characteristics of different transmission and network technologies into a consistent network layer service. A VTOC for layer 3 is presented in Figure A-8

#### A.1.2.3 Link & Physical Layers - Layer 2 & 1

Both of these layers were implemented in hardware as described in previous sections.

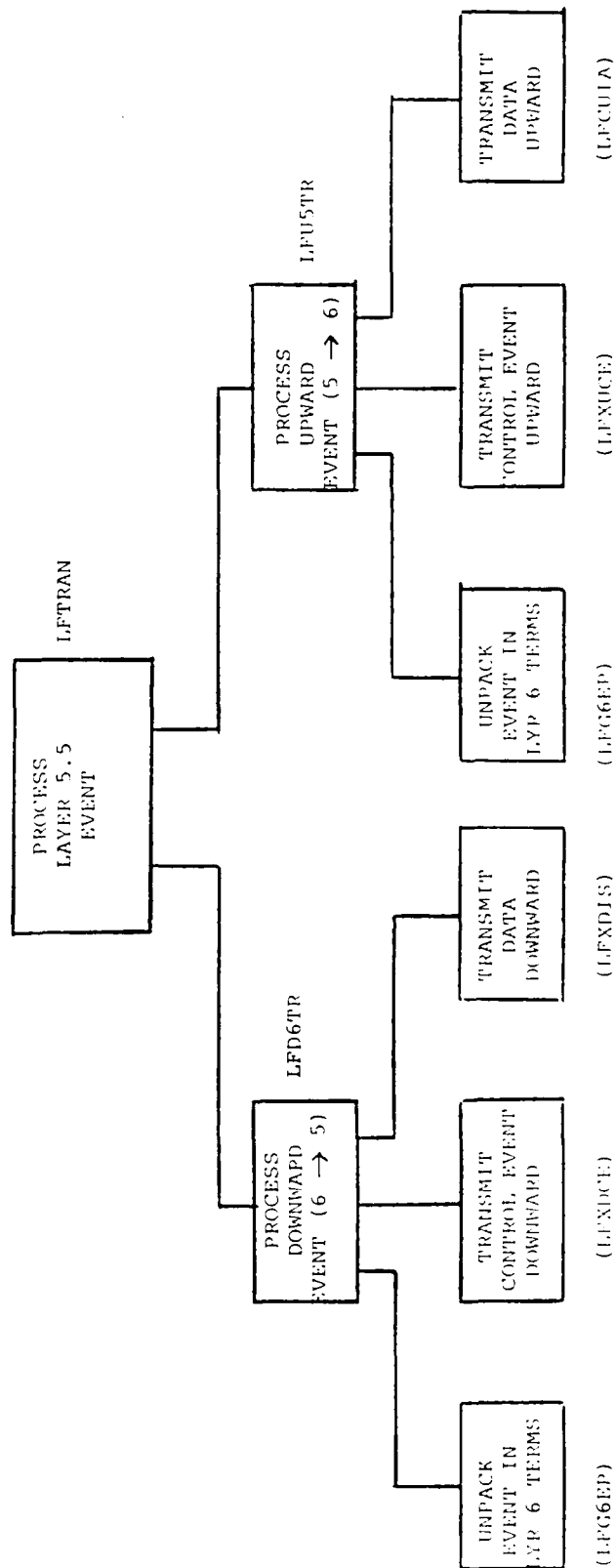
#### A.1.3 Protocol Translation Routines

In the CCITT Recommendation, protocols are described as between two terminals on the same layer; the inter layer communication is usually only implied. The validation system protocol modules follow this organization; they communicate with their own layer on the other side of the transmission in terms of the protocol for their own layer, but also perform the translation/conversion between it and the next higher layer. This translation/conversion is done by the translator routines (half layer routines) for each protocol layer. These routines thereby allow the protocol processing routines to be concerned with only the implementation of the protocol for that layer and not any of the repackaging tasks required for the interlayer communication. Shown in Figures A-9 through A-12 are VTOCs for the half layer translation routines.



LAYER 6.5 VIOC

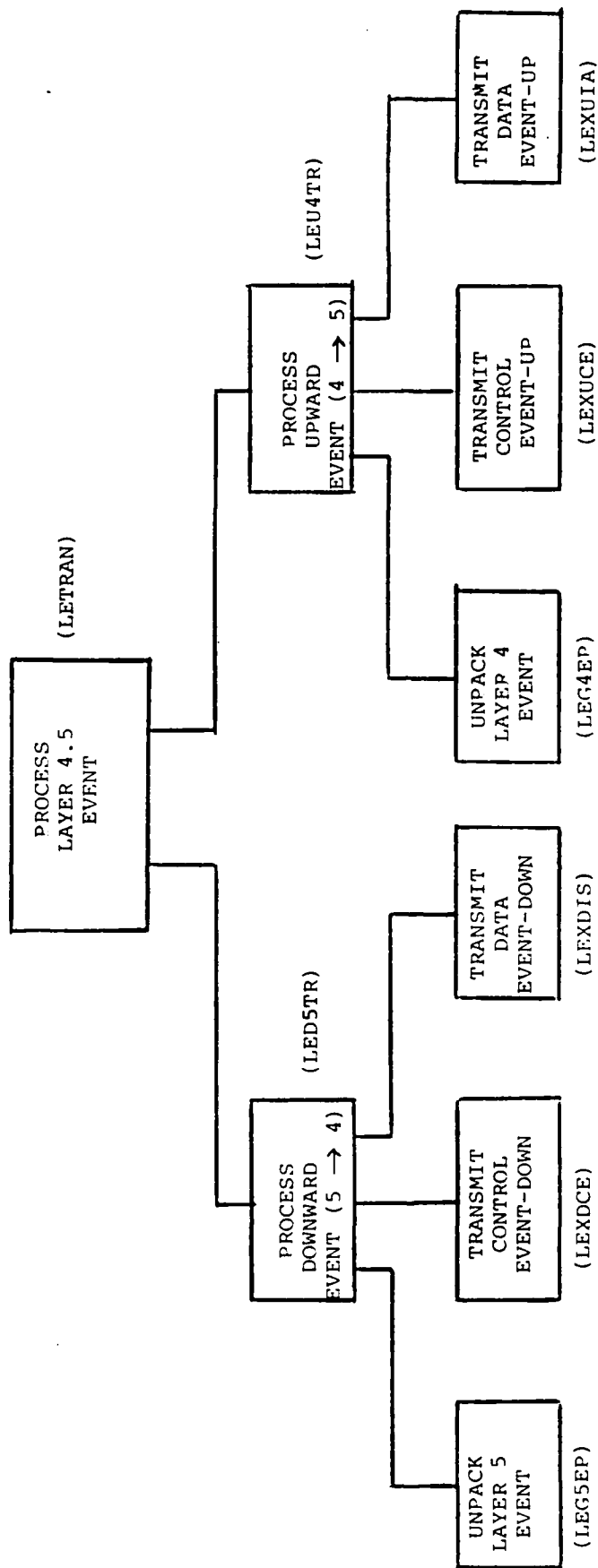
FIGURE A-9



LAYER 5.5 VTOC

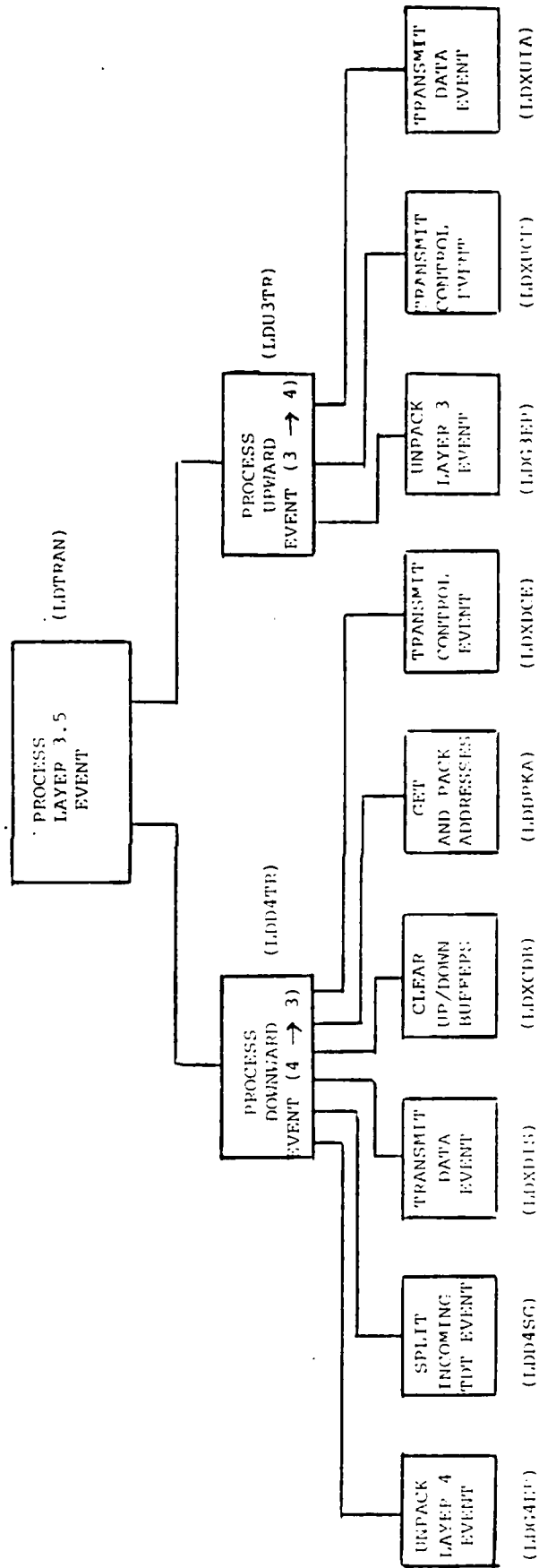
FIGURE A-10





LAYER 4.5 VTOC

FIGURE A-11



LAYER 3.5 VTOC

FIGURE A-12

## A.2 Event Processing Routines

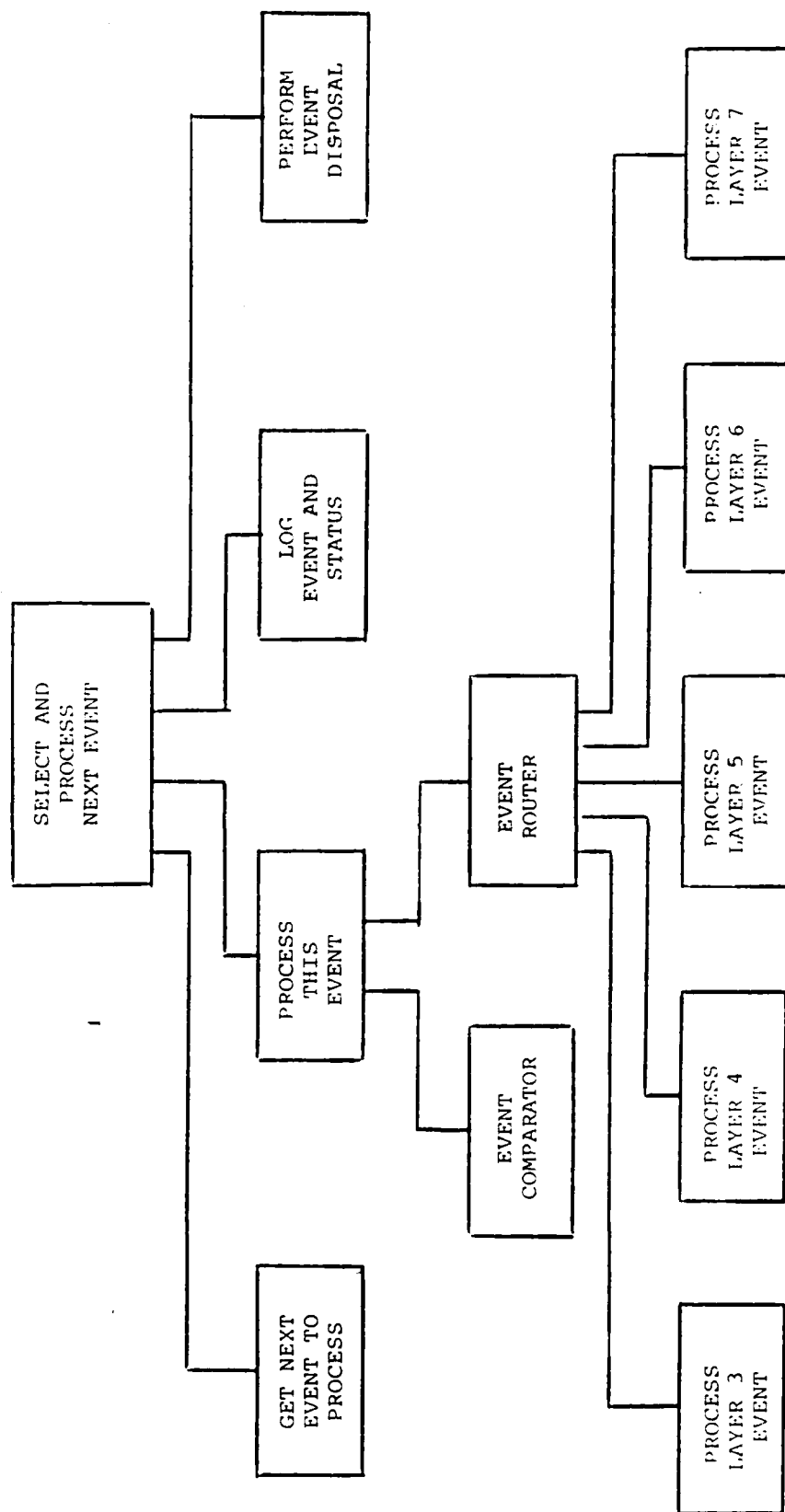
Within the Group 4 Validation system all activity or communication is initiated and processed as an event. The event processing routines control and monitor all system activity. These routines insure that all events within the validation system are handled in the proper sequence and also perform the comparisons necessary for validating the Group 4 UUT.

Shown in Figure A-13 is a VTOC of validation system event processing routines.

## A.3 Test Select Configuration Routines

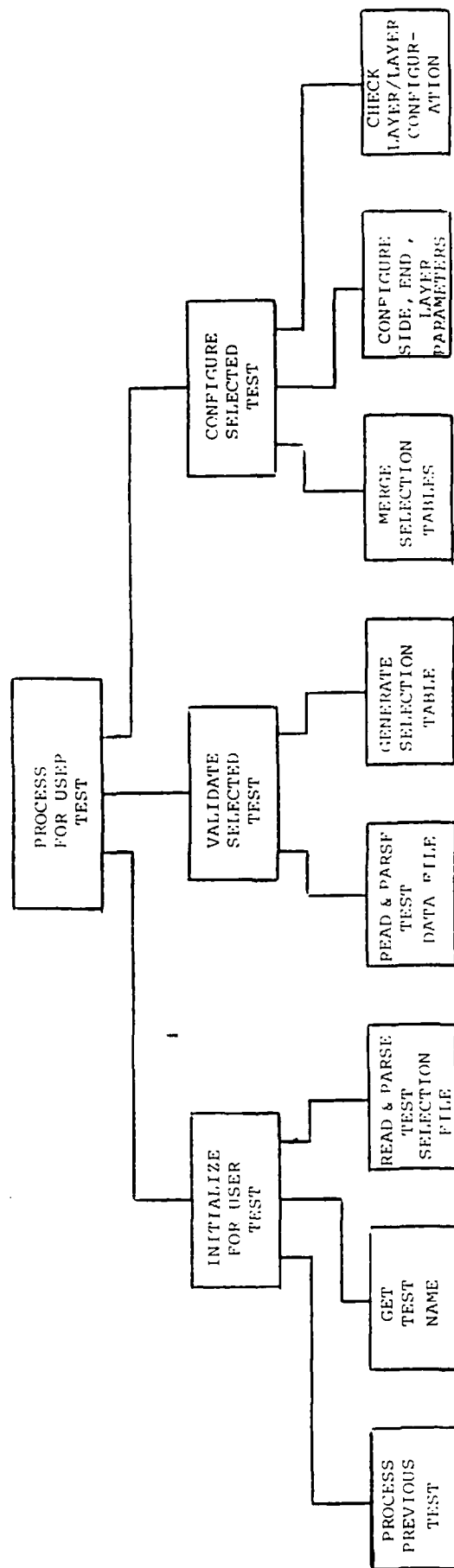
The test selection and configuration routines are those routines that parse, process, validate and configure the validation system for the currently selected tests. All parameters for each of the protocol layers are validated and established with an appropriate diagnostic message issued if a parameter error is detected. The parameters may be a complete parameter set defining the parameters for all layers or a single parameter change to a currently existing parameter set. In this way a base validation test can be executed and then modified by a single parameter change to a particular layer and then re-executed. The validation system is also capable of repeating the currently selected test if requested by the operator.

A VTOC describing the parameter processing is shown in Figure A-14. The test selection routines basically read and parse the currently selected test, validate the test parameters, merge the new



VTOC OF VALIDATION SYSTEM EVENT PROCESSING ROUTINES

FIGURE A-13



PARAMETER PROCESSING VTOC

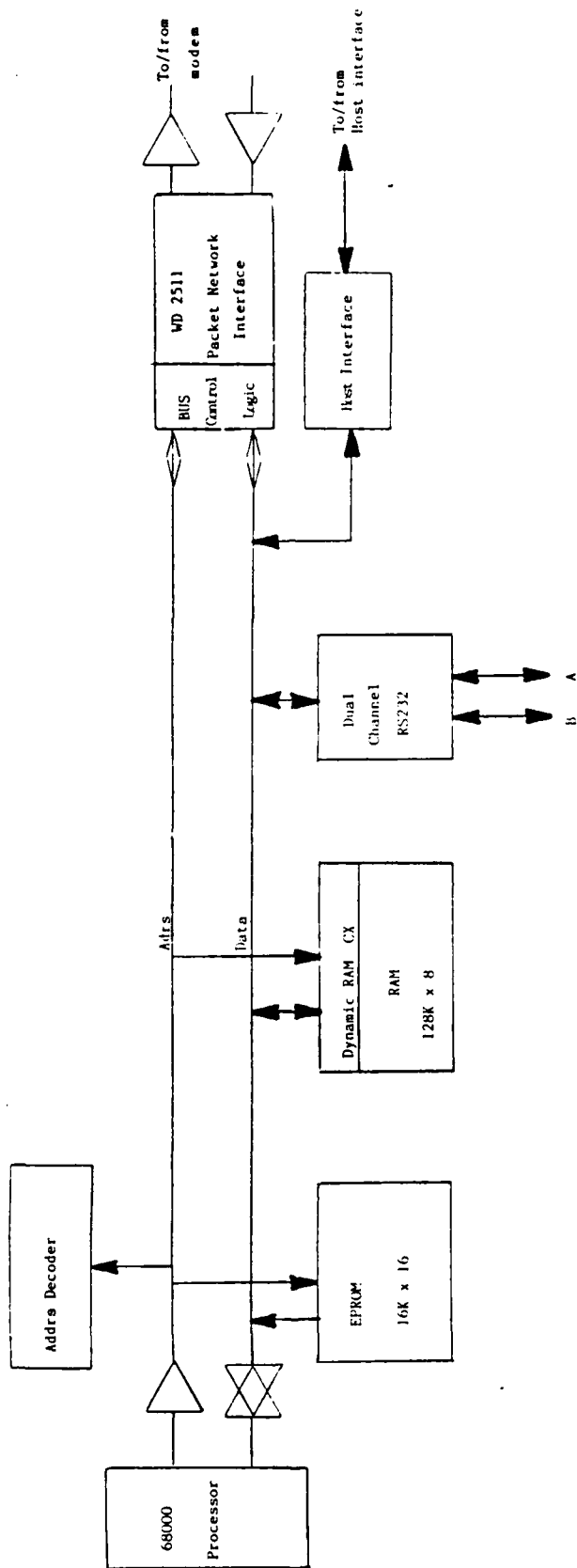
FIGURE A-14

parameters with any currently existing parameters and then configure the validation system for the test.

## APPENDIX B

### Validation System Hardware Description

This section is the block diagram for the Packet Data Interface (PDI).



PACKET DATA INTERFACE  
Front end block diagram



## APPENDIX C

### TEST SELECTION AND PROCESSING SYNTAX

#### C.0 Test Select and Test (processing) Files

In order to perform a test, two "files" are needed; the test file itself and the test select file. The former contains (or specifies the source of) the test "script" (run data), preceded by its "standard" (default) configuration parameters. The test select file, on the other hand, provides parameter values which supplement or override the default values provided in the test file itself. The following sections describe the organization of and the syntaxes used in these two files, and how they are used. Both files are sequentially formatted (i.e., ASCII) files with a maximum record length of eighty (80) characters, of which only the first 72 are interpreted, columns 73-80 being reserved for sequencing and similar purposes..

#### C.1 Test (processing) File Organization

The test file itself is divided into two parts: its default parameter section and its run data. The parameter section is headed by a record of the form:

columns 1-16: \$TEST\_PARAMETERS  
columns 17-32: version identification  
columns 33-72: (not used)

The version identification permits several different sets of default

parameters for a given test to be "stacked" on a single file for convenience; see C.6.1. ("TEST=" parameter) regarding specific version selection.

The body of the test parameter selection uses test select syntax (C.3.) to supply the default configuration parameters for the test. While this same syntax is also used in the the test select file (C.2.), certain parameters and forms are applicable only in the test file itself; these are described in C.5. below.

The "script" portion of the test file is headed by a record of the form:

columns 1-16: \$TEST\_RUN\_DATA (left justified)  
columns 17-32: version identification  
columns 33-72: (not used)

As in the parameter portion, the version identification permits stacking of multiple scripts on a single file; see C.6.1. for version selection methods.

The body of the script uses test processing syntax (C.7.). The script consists of a series of processing commands which initiate, control, and terminate the performance of a test. As presently implemented, only the "/QUEUE" (insert an event, including data if any, into the event queue), "/NOTE" (provide information to operator), and "/END" commands are handled, although much of the design and infrastructure implementation have been done to handle conditional and iterative processing, as well as handling commands and data from external devices.

## C.2. Test Select Organization

To select a test, the operator may either use a previously prepared test select file or enter the necessary test select parameters on line. If the test select parameters are "on file", they must be headed by a record of the form:

```
columns 1-16: $TEST_SELECT          (left justified)
columns 17-32: version identification
columns 33-72: (not used)
```

As in the test file, the version identification can be used to distinguish between different test select "decks" stacked on the same file. The operator chooses a specific version by including its identification with the file description entered in response to the test selection request; see C.6.1. for the form used.

Regardless of whether the test selection "file" is "canned" or entered on line, its body follows the test select syntax described in C.3. below. Certain parameters and forms, however, are applicable only to test select files; these are described in section C.6.

## C.3 Test Select Syntax

In general, the body of a test select file (or the parameter portion of a test file) consists of a list of parameters separated by commas (or one or more blanks) and terminated by a dollar sign. The list may occupy one or more records (lines); the line-to-line boundary has the effect of a blank. These parameters may be parenthesized by groups, either for readability purposes or localization control (see C.4. below).

To aid in the description of the syntax, especially at the low level, certain metasyntactic conventions have been adopted in both this section and those following. These conventions are:

- Uppercase letters, digits, and punctuation generally (except for square brackets and ellipses) represent themselves; i.e., they are "fixed" representations to be used exactly as written.
- Lowercase letters represent "variable" (generally alphanumeric) syntactic entities to be "filled in", or whose representation is elsewhere described (i.e., "non-terminals").
- Square brackets ([ ]) set off optionally omitted parts of the syntactic item in question.
- An ellipsis (...) following an optional (square-bracketed) part indicates zero or more repetitions of that part.
- blanks are used freely to enhance readability.

For example, the form described by:

PARM = val[,val]...

covers all of the following instances:

PARM = A

PARM = R, S

PARM = U,V,X,Y,Z

but does not cover either of the following:

VAL = A (must be "PARM")

PARM (nonoptional "=val" missing)

### C.3.1 High-level Syntax

The overall form of a test select "deck" is:

```
paramlist $
```

where the form of "paramlist" is:

```
[parameter[,parameter]...]
```

i.e., a possibly empty list of parameters as noted above. Note that the empty case is valid only for the test parameter portion of a test file; a test select file must contain test file identification, at least. These definitions are given here for recursive use in defining parenthesized parameter values (see C.3.4.).

### C.3.2 Parameter Format

The form of a single parameter is:

```
[paramname =] paramvalue
```

The "keyworded" form (where "paramname =" is present) is normally used where a specific parameter value is to be supplied. For example, assuming prior localization to layer 5:

```
CTlT=45
```

sets the T.62 (layer 5) Tl (inactivity) timeout period to 45 seconds.

The "positional" form (where the name is omitted) is chiefly used in those cases where the parameter concerned can have only two possible values (e.g., "on" or "off"), and the value can consist of the parameter name itself, possibly preceded by a confirming or denying sign. For example, in the same context as above:

```
QULL
```

causes the layer 5 module to produce "long" (3-octet) length

indicators regardless of the length to be represented.

Certain specific parameters are actually positional; i.e., their effect depends on their position in the test select deck as a whole. For example, in the test select file, the test identification parameter is usually keyed with the parameter name "TEST", but in the absence of such a parameter, the first parameter in the file, if not keyworded, is assumed to be the test identification parameter.

#### C.3.3 Keyword (paramname) Format

A keyword must consist of a letter followed by 0-7 letters or digits. Within a keyword, case is not significant; all letters are handled as if in uppercase; for example, the keywords "key", "KEY", and "Key" are equivalent, all being represented by the capitalized form "KEY".

Keywords supply the "names" of parameters, which can generally be divided into three classes: global (e.g., test identification), local (e.g., CT1T above), and "localizing" parameters, which restrict the effect of following local parameters to specific sides, ends, and layers.

#### C.3.4 Parameter value (paramvalue) format.

Parameter values may assume any of seven forms:

[sign] anvalue

'string'

"string"

(paramlist)

\*

&

[] (i.e., null)

In the first form above, "anvalue" is a string containing digits, letters, underscores, or a mixture of them; its maximum length is 72 characters. As in keywords, case is not significant; all letters are treated as if in upper case. If only digits are involved, it is interpreted as a decimal value; otherwise its interpretation is dependent on the specific parameter concerned.

The optional "sign" may be either a plus(+) or a minus(-). The latter has the effect of negating the value "anvalue", either arithmetically (if the value is numeric) or logically. For example, in the context of the second example in C.3.2 above:

-QULL

has the effect of "turning off" the "long length indicator" force implied by the unsigned form of the value. A plus(+) sign, on the other hand, just "affirms" the "positive" sense of the value; its use is equivalent to the omission of the sign entirely.

The delimited string forms (second and third in the list above), may contain any printable ASCII characters, with the restriction that an occurrence of the delimiting character itself within "string" must be represented by a pair of the character with no other characters intervening.

For example, the value:

That's all folks

could be represented by either:

'That''s all, folks'

or:

"That's all, folks"

In this form, case is significant; "a" is not equivalent to "A".

The delimited form is generally used when the value must contain blanks or other special characters. For example, this form is required in the test identification parameter on the test select file, where colons, periods, etc. are needed to satisfy file description directory syntax, as well as to set off imbedded version identification.

The fourth (parenthesized) form is used to group related parameters for documentation purposes, and to limit localization effects (see C.4. below). If the value of a keyworded parameter is of this form, and the first parameter in the parenthesized list is not keyworded, the group keyword is assumed to apply to that first grouped parameter. For example:

LYR=(5,QULL)

is equivalent to:

(LYR=5,QULL)

Other uses of group keywords have essentially no effect except for documentation purposes and certain uses in the test parameter section of the test file (see section C.5.).

The last three forms ("\*", "&", and null) are applicable only to the test select file; their use is described in section C.6.2. Generally, they specify the choice of either the last used (repeat) or standard (default) value for the parameter in question.



#### C.3.5 Use of blanks.

Generally, blanks may be used freely adjacent to other "breaks" to enhance the readability of the selection list. One or more blanks may also be used to separate parameters themselves; when so used, they have the effect of a comma. However, blanks may not be embedded in keywords or in the "anvalue" portion of the first parameter value form (C.3.4. above), although they may be used between the sign and the value proper. This restriction further implies that keywords and "anvalues" may not be split between lines. To limit the possible lexical "states" at the end of a line, splitting of values in the delimited string form is also prohibited.

Blanks used adjacent to "breaks" are not significant; if a set of blanks is used as a break, its only significance is the separation function itself. However, blanks imbedded within delimited string values form a significant part of that value.

#### C.4 "Localizing" Parameters

As has been noted elsewhere, the "core" of the test execution subsystem consists of sets of modules for each OSI layer plus translation half-layers between. The set for a specific layer or half-layer can operate differently for each side and end combination involved in the test as configured. To permit this, it is necessary to restrict the effect of "local" parameters to the specific side/end/layer combinations used, as well as to limit the scope of layer-specific parameter names to avoid conflicts. The "localizing" parameters perform this restriction function by selecting the specific sides, ends, and layers affected by following layer-specific parameters.

Test select processing (for both the test select file and the parameter portion of the test file) not only stores the names and values of the parameters specified, but also notes which side/end/layer combinations are affected by them. Initially (at the beginning of the deck), all side, end, and layer selections are "off" (for processing context-independent "global" parameters); to make "local" parameters effective at least one side/end/layer combination must be "turned on". This function is performed by the "SID", "END", and "LYR" parameters acting independently.

The effect of one of these "localizing" parameters persists until "switched" off or to another combination; its effect applies also to any parenthesized parameter groups "nested" following it at the same level. However, the effect of a localizing parameter within a parenthesized group "dies" at the end of the group; for example, in

the following parameter list fragment:

....,(LYR=5,QULL),PABC,...

the effect of QULL is localized to layer 5, but localization of PABC is dependent on that in effect prior to the parenthesized group.

#### C.4.1 Side Selection Parameters.

The form of a side selection parameter is:

SID=sidv

where "sidv" chooses the effective side(s) for subsequent local parameters as follows:

<u>Side(s) to be selected</u>	<u>Alternative "sidv" values</u>
Test (UUT) side	TEST, UUT, U, or 1
Control emulation side	CTRL, EMUL, E, C, or 2
Both sides	BOTH or B
Neither side (turnoff)	0 (or none of the above)

#### C.4.2 End Selection Parameters

The form of an end selection parameter is:

END=endv

where "endv" chooses the effective end for subsequent local parameters as follows:

<u>End(s) to be selected</u>	<u>Alternative "endv" values</u>
Tester (near) end	NEAR, TEST, N, T, or 1
UUT (far) end	UUT, FAR, U, T, or 2
Both ends	BOTH or B
Neither end (turnoff)	0 (or none of the above)

### C.4.3 Layer Selection Parameters.

This parameter may be used to select either a single layer, no layer (turnoff), or a range of contiguous layers.

The form of a single layer selection is:

LYR=lyrv

where "lyrv" indicates the layer to be selected. The form of the layer range selection is:

LYR=lyrva\_lyrvb

where "lyrva" and "lyrvb" indicate the (inclusive) ends of the range specified. The various layers and half-layers are represented as follows:

Layer	Description	Alternative "lyrv" values
0.5	physical near<->far connection	A, H, or 0H
1.0	physical layer	PHY or 1
1.5	physical<->link	B or 1H
2.0	link layer	LNK or 2
2.5	link<->network	C or 2H
3.0	network layer	NTW or 3
3.5	network<->transport	D or 3H
4.0	transport layer	TRN or 4
4.5	transport<->session	E or 4H
5.0	session layer	SSN or 5
5.5	session<->presentation	F or 5H
6.0	presentation layer	PRS or 6
6.5	presentation<->application	G or 6H
7.0	application layer	APP or 7
7.5	user<->application interface	U or 7H
	(none) (turnoff)	0 (or none of above)

For example:

LYR=A\_3

selects layers 0.5 through 3.0 inclusive.

#### C.4.4 Layer Configuration Parameters

While this particular parameter is strictly a "local" rather than a "localizing" parameter, it is described in this section because its semantics is shared by all layers, as opposed to the majority of local parameters whose names and functions are peculiar to specific layers. The function of this parameter is to describe what layers are "present" (on a given side and end) and how they are "accessible" to the event queue in the given test performance. A given configuration parameter may specify, for example, that layer 3 is "accessible" from higher layers (3.5 and above) but not from below (2.5 and lower), because the latter layers are "buried" in an X.25 "chip". The semantics of this particular parameter was made common to all layers in order to facilitate consistency checks between the configurations in adjacent layers.

The form of the layer configuration parameter is:

CONFIG=cfgv

where "cfgv" chooses the appropriate configuration option:

Layer configuration -----	Alternative "cfgv" values -----
Protocol or translator, fully accessible	PROTOCOL, TRANSLAT, PRTCL, TRNSLT, PR, TR, P or T
Interface accessible from higher layers only	UINTF, U, HINTF or H
Interface accessible from lower layers only	LINTF, L, BINTF or B
Inaccessible layer (buried in hardware)	MINTF or MI
Not present (turnoff)	(negation of any of above)

## C.5 Parameters Specific to Test File

Certain parameters and forms are applicable only to the test parameter portion of the test file. These parameters provide a means for the test writer to regulate the use of the test select file to supplement and/or override "standard" test parameters.

### C.5.1 Override Prevention.

By the use of the `FIXED` parameter keyword, the test file can prevent the test select file from overriding specified parameter values in the test file. This keyword is used as follows:

```
FIXED=(paramlist)
```

This usage causes the diagnosis of an error (`INVALID SELECT; FIXED BY TEST`) if the test select file attempts to supply a value for any of the parameters in the parenthesized "paramlist" above. The error will be diagnosed separately for each "fixed" parameter and each side/end/layer combination "covered".

The `FIXED` parameter does not affect the "localizing" parameters (C.4. above), but works in parallel with them. To illustrate this, in the following fragment:

```
...,FIXED=((LYR=5,-QULL),PXYZ=1),...
```

overrides of the `QULL` parameter are prohibited for layer 5 only, but the combinations for which `PXYZ` overrides are prevented depends on the localization in effect (if any) prior to the `FIXED` parameter in the test parameter list.

The effect of the `FIXED` parameter persists within the

parenthesized list it keys, including any further lists nested within it, unless "switched" by a REQUIRED or OPTIONAL parameter in the list.

#### C.5.2 Requirement for Supply of Value

The REQUIRED parameter demands that the test select file supply values for specific parameters in the same way that the FIXED parameter prohibits it. It is used similarly:

REQUIRED=(paramlist)

An error will be diagnosed (REQUIRED USER SELECTION ABSENT) if the test select file does not supply a value for each of the parenthesized parameters. Similarly to the FIXED parameter, values must be supplied (separately or together) for each side/end/layer combination in effect (within the test file) for the parameter in question.

The parameters within the REQUIRED parenthesized list are normally written as keyworded with null values; for example:

...,CONFIG=,...

to emphasize that values must be supplied. This is an exception to the use of this form only in test select files.

#### C.5.3 Optional Override Restoration.

The OPTIONAL parameter is used to "turn off" the effect of either the FIXED or REQUIRED parameter:

OPTIONAL=(paramlist)

Its use restores the "normal" mode of operation, in which either the test parameters or the test select file may supply the value for a given parameter, the test select value being used if both do. The

persistence of its effect is similar to that of the FIXED and REQUIRED parameters, as is its interaction with localizing parameters.



## C.6 Parameters Specific to the Test Select File

Certain parameters and value forms have meaning only within the test select file. These are the test identification parameter and the indirect value references.

### C.6.1 Test Identification Parameter

This parameter is used to specify the test file to be used. Its form is one of the following:

```
[TEST=]'[filed sc][,U=unit][,V=version]'  
[TEST=]"[filedesc][,U=unit][,V=version]"  
[TEST=]*  
[TEST=]&
```

The keyword TEST may be omitted only if the parameter is the first in the test select file; in this case, it must not be within a parenthesized list. The remaining parts are used to identify the test file by directory information (or as the last test performed), supply a specific unit number to read it, and to select a specific version of it if required.

The "filedesc" part must be the first within the delimited string, but the "unit" and "version" parts may be in either order. The handling software will also permit (but not interpret) other comma-separated parts (to allow for future enhancements) anywhere after the "filedesc" part. Trailing commentary is also allowed, indicated by a preceding exclamation mark(!).

#### C.6.1.1 File Descriptor

The file description part is normally required, but may be omitted if the unit part (C.6.1.2) is sufficient to identify the test "file". The form of "filedesc" must be such that it can be used as a "filename" specifier in a Fortran OPEN statement preparatory to reading the test file.

The use of the asterisk (\*) or ampersand (&) forms indicate that the test to be selected is the one just performed. The asterisk specifies that the test is to use parameter values as last performed (except for specific values in the test select file); the ampersand, that "standard" parameter values (as specified in the test file itself) are to be used.

#### C.6.1.2 Unit Specifier.

The unit specifier may be omitted except when the file descriptor is absent, or when a specific unit number is to be used to read the test file. Its value "unit" must be in the form of a decimal integer suitable for use as a Fortran unit number in OPEN and READ statements. Omission of this parameter causes a program-defined default unit number to be used for reading the test file.

#### C.6.1.3 Version Specifier

This specifier is used to identify specific versions of both the test parameters and the test run data parts on the test file. Its value "version" must be an alphanumeric string one to sixteen

characters in length. Its purpose is to select one of several different test parameter or run data "decks" stacked on a single file. If the version specifier is omitted from the test identification parameter, the version portion of the deck header (columns 17-32; see C.1.) is ignored; i.e., the first version found will be selected.

#### C.6.1.4 Test Select File Identification

The same options used in the test select file to identify the test file may be used by the operator to identify the test select file when requested to do so by the validator, although with adaptations to on-line entry. Specifically, the explicit form of test select file identification becomes:

```
[filedesc][,U=unit][,V=version]
```

where the keyword and the string delimiters have been dropped. For this usage, "filedesc" identifies the test select file rather than the test file; the uses of the "unit" and "version" parameters, however, completely parallel those in test identification.

#### C.6.1.5 NEW, REPEAT, and DEFAULT Parameters

While the NEW, REPEAT and DEFAULT parameters are syntactically separate from the test identification parameter, their function is essentially inseparable from it. Their form is as unkeyworded parameters:

NEW

REPEAT

DEFAULT

One of them may be used immediately following the test identification

parameter, i.e. as the second test select parameter; the test identification must be explicit. Their use is to "force" the read of the test file, or to permit inclusion of the explicit test file identification even if the last performed test, or its "standard" version, is to be re-run.

The NEW parameter specifically forces the identified test to be read, even though the same test was just performed. The REPEAT parameter permits the inclusion of the test identification for repeating the test just run (identifications must match) for logging purposes. For example, if the test just run was identified as A:TESTABC.TST, then:

```
TEST='A:TESTABC.TST',REPEAT
```

is equivalent to (and better for logging purposes than):

```
TEST=*
```

The DEFAULT parameter permits similar inclusion of the test identification when the "standard" test is to be repeated.

#### C.6.2 Indirect Value References

In the test select file, parameter values may be indirect references to those used as "standard" for the test involved or to the values used in the test just performed. The form of such an indirect reference is (from C.3.3. above) one of:

\*

&

[] (null)

The asterisk (\*) indicates that the last used value for the parameter in question is to be "picked up"; it is applicable only if the same

test is to be performed as the same as was last done. The ampersand (&) specifies that the "standard" value (from the test parameters) is to be used. Finally, the null form "goes with the test"; it has the effect of "\*" if the test is a repeat, the effect of "&" if "default", and is ignored if the test is "new". For example, in the following fragments:

```
TEST=*,..., PARA=&, PARB=, PARC=*,...
```

PARA assumes its "standard" value, but both PARB and PARC assume the values used in the last performance of the test, PARC explicitly, but PARB because the test itself is a repeat.

Because of localization, a single "\*", for example, may cause several different values to be picked up. For example, if the localization of a single parameter for which "\*" is specified covers several layer/side/end combinations, a different value may be picked up for each combination.

### C.7 Test Processing (run data) Syntax

The run data (script) portion of the test file contains the test processing commands which initiate, control, and supply data during test execution. Generally, the script consists of a series of commands, each of which comprises one or more records. The script is terminated by a "sentinel" record which has a dollar sign (\$) in column 1.

The first (header) record of each command has the following format:

```
column 1      : /  
columns 2- 8: command  
columns 9-72: dependent on specific command, but normally  
              divided into eight 8-column fields.
```

The remaining records of the command have a blank in column 1, but their format is otherwise dependent on the specific command.

With the exception of the "/NOTE" command, all the commands so far implemented do not depend on the column 1 convention above to "complete" their definition; they "know" when they're "finished". As a result, processing these commands which inadvertently have too few or too many continuation records will result in appropriate diagnostics. It is felt that this is a worthwhile "fail-soft" practice to continue.

While provision has been made, and the infrastructure built for, many more commands involving conditional execution, iteration of script portions, and transmission of data or processing commands from

remote devices, only the /QUEUE (insert event into the queue), /NOTE (provide information and "breakpoint" to operator), and /END (terminate test processing) commands have been implemented at the present time.

#### C.7.1 General Notes on Script Processing.

All script processing is done during the test execution phase. When the user-application layer module is invoked by the poll event provided for that purpose, it in turn invokes the test processing module which "reads" the script; this module then processes one or more commands; any events resulting are then processed by the interface module. This mechanism is first used to insert "seed" events into the queue, and later to supply further "external" control and data. Finally, the /END command "terminates" the script (but not the test), and allows the test to work itself out.

#### C.7.2 Event Insertion

The /QUEUE command is used to build and insert an event (with or without data) into the event queue. While this event is normally "aimed" at the user-application layer interface (layer 7.5) at the tester (near) end, the /QUEUE command may build events targeted toward any layer/side/end combination.

The form of the /QUEUE command's header line is:

```
columns 1- 8: /QUEUE
columns 9-16: [procopt]
columns 17-24: [evcode]
```

columns 25-72: [fldspec]...[dataspec]

Each field specification "fldspec" occupies a separate eight-column field. The data specification "dataspec", if used, occupies one or more eight-column fields. These specifications need not be left-justified in their fields, and trailing blanks in the fields are ignored, except in the "middle" of the "dataspec". Except for the "dataspec", which must be last if used, the specifications may be in any order.

#### C.7.2.1 Processing Option

The processing option field "procopt" specifies whether the test processor will return control to the event queue (via the user-application layer interface module) after building and inserting the event. The form of "procopt" is one of:

PROCESS

HOLD

The first (PROCESS) form causes immediate return of control after event insertion; the second (HOLD) allows the test processing module to continue reading the script. If the "procopt" field is blank, PROCESS is assumed.

#### C.7.2.2 Event Code

This field specifies the event code to be used in the event. Its form is as follows:

[eg+]evc

in which the use of the optional event group indicator "eg" permits specification of the code relative to a group "base"; the omission



permits specifying the code in absolute terms.

If the optional "eg" is used, it must be a two-character event group indicator occurring in the following list:

Indicator	Description	Implied Layer*
OH	layer 0.5 local	0.5
10	layer 1.0<->0.5	1.0
11	layer 1.0 peer & local	1.0
12	layer 1.0<->1.5	1.0
1H	layer 1.5 local	1.5
..	...	...
76	layer 7.0<->6.5	7.0
77	layer 7.0 peer & local	7.0
7U	layer 7.0<->7.5(U)	7.0
7H	layer 7.5(U) local	7.5

\*(see C.7.2.3.1.)

The event code portion "evc" is interpreted as a decimal integer.

The event code to be used in the event is calculated by adding the value of "evc" to that for the group base (if any) or to zero if no group is specified. The group base value is the event code for the "zeroth" event in the group, i.e., one less than the lowest event code of the group.

If columns 17-24 are blank, the event code of the user-application layer poll event is used.

#### C.7.2.3 Other Event Field "fldspec" Specifications

The field specifications "fldspec" are used to populate the remaining "non-pointer" fields of the event. Each "fldspec" is one of:

lsespec	(layer/side/end)
dtpspec	(direction/target/priority)

auxspec	(auxiliary)
sscspec	(state/substate change)
clkspec	(clock time)

A single /QUEUE command can contain no more than one of each kind of "fldspec".

#### C.7.2.3.1 Layer, Side, and End.

The form of "lsespec" is either of:

L=lse

L=?

where "l", "s", and "e" are separate one-character indicators, either alphanumerics indicating specific layers, sides, and ends, or asterisks (\*) which call for the default values.

Valid layer indicators are "A", "1", "B", ..., "G", "7", or "U", where the digits represent "full" layers and the letters the translating "half-layers". The default (if "l" is "\*" or "lsespec" is omitted) is the implied layer for the event code specified (see C.7.2.2. table).

Valid side indicators are "U" (UUT, tester) or "C" (control emulation). The default is "U".

Valid end indicators are "N" (near, tester) or "F" (far, UUT). The default is "N".

For example, the specification:

L=5C\*

indicates the target to be layer 5 on the control emulation side at the "near" end (the last by default).

The second form of "lsespec" (with the question mark) causes the operator to be prompted to enter the layer, side, and end in the "lse" form (without the keying "L=").

#### C.7.2.3.2 Direction, Target, and Priority ("dtpspec").

The form of "dtpspec" is either of:

H=ntp

H=?

where "d", "t", and "p" are separate one-character indicators of the event direction, target module class, and handling priority respectively. Functionally, these indicators may be in any order; the indicator value itself indicates which of the three fields it specifies. If no value specifying a field is used, the field assumes its default value.

Valid direction indicators are "U" (upward) or "D" (downward); downward is the default direction. Valid target classes are "K" (comparator or traffic cop module), "P" (protocol or translator module), or "I" (interface module); default is "K".

Valid priority indicators are "X" (express), "N" (normal), or "T" (deferred timeout); "N" is the default.

For example, the handling specification:

H=TP specifies a downward direction (by default), a protocol module target, and deferred timeout priority.

The second form of "dtpspec" causes the operator to be prompted for the "ntp" information, similarly to the corresponding "lsespec"

form.

#### C.7.2.3.3 Auxiliary Field

The form of the auxiliary field specification "auxspec" is either of:

A=hh

A=?

where "hh" is one or two hexadecimal digits whose value will be used to populate the auxiliary field of the event. This field defaults (if no "auxspec" is used) to zero. The second form causes the operator to be prompted for this value as in the explicit form.

#### C.7.2.3.4 States and Substate Change.

The form of "sscspec" is either of:

[S=]nn[<oo[/cc]]

S=?

with the constraint in the first form that not both the keying "S=" and the "less than" (<) part be omitted. In the first form, "nn", "oo", and "cc" are each one or two hexadecimal digits, whose values are to be used to populate the new state, old state, and substate change fields of the event respectively. In the second form, the operator is prompted for this information, to be supplied as in the first form, omitting the keying "S=". Omitted fields default to zero.

#### C.7.2.3.5 Clock Time Specification

The form of the clock time specification "clkspec" is either of:

@[+]hhmmss

@[=]?

In the first form, the use of the plus (+) indicates that the value of the following "hhmmss" is to be added to the current clock time (i.e., is incremental); the omission of the plus indicates that "hhmmss" is to be interpreted absolutely. "hhmmss" must be a string of digits representing hours, minutes, and seconds. If less than six digits, it is, in effect, right justified and padded on the left with zeroes.

For example:

@+230

indicates that the clock time to be used in the event is the current clock time plus two and a half minutes.

The second form calls for operator input of the absolute or incremental time to be used in the event; the form used should follow the first form above (less the keying "@").

#### C.7.2.4 Data Specification

The data specification on a /QUEUE header may completely contain the data, indicate that the data is on following continuation records, or indicate that the operator is to supply it. The form of the data specification is one of:

D='string'

D="string"

D=&

D=?

In the first two forms, the data for the event consists of the delimited string. Within the string, the usual rules are followed for representing the delimiter character itself: it must be doubled without intervening characters. The third form indicates that the data itself are on one or more continuation records following, while the last form causes the operator to be prompted to supply the data. If the /QUEUE header does not contain a data specification, the event carries no data package.

In the following paragraphs, the expected form of the data, either on continuation records or as operator input, will be detailed. Alternate methods capable of supplying "non-ASCII" data will be described for both modes of input. The approach will be to describe simple "ASCII" continuation and note the differences if "non-ASCII" data and/or operator input are involved.

#### C.7.2.4.1 ASCII Data Continuation

This form may be used if the data consists entirely of printable ASCII characters. It may also be used for those records containing such data in multirecord data "packages".

For a /QUEUE command in which the "dataspec" is "D=&", the data which are to form the event data "package" are specified on one or more data continuation records following. In these records, column 1 must be blank, but columns 2-72 may be used freely for the data and its delimiters.

Each such continuation record carries a "piece" of the data;

these "pieces" will be concatenated in the same order as the continuation records to form the event's data package. On each record, the data "piece" must be delimited on both sides. On the first continuation record, the left delimiter must be either an apostrophe (') or a quote ("); this delimiter must be matched by the right delimiter on the last continuation record. All other delimiters (on multirecord "packages") must be ampersands (&).

Each "piece" may be from 1 to 69 characters long (excluding delimiters); except for the "piece" and its delimiters, all characters in the record (in columns 1-72) must be blank. In other words, the first and last non-blank characters of the record must be the delimiters.

Within each piece, the start/end delimiter (apostrophe or quote) can be itself represented by doubling it in the usual fashion. No special methods are used for representing ampersands; its delimiting function is implied by its use as the first or last non-blank character on a record.

Let us give a few examples to illustrate the form. Assume that the "data package" is to be:

That's all, ladies & gentlemen!

To represent it in a single record, we could use (assuming a starting column other than 1):

'That''s all, ladies & gentlemen!'

or alternatively:

"That's all, ladies & gentlemen!"

For a multirecord representation, we could use:

```
'That&
&'s all, ladies &&
& gentlemen!'
or using a "staggered" representation:
"That's all&
&, ladies & gentlemen!"
```

#### C.7.2.4.2 Non-ASCII (over/under) Data Continuation

In order to represent octets which do not correspond to printable ASCII codes (or to represent any octet, for that matter), an "over/under" technique is used in which two succeeding records represent a single data "piece", "special" octets being represented by two hexadecimal digits in corresponding columns of the two records, and "ordinary" (ASCII) octets being similarly represented by a pair in which one of the records has a blank in the column concerned.

A special continuation character - the vertical bar (|) - is used to delimit the first of the over/under pair and to indicate that such a pair is beginning. The second (the "under" part) of the pair is delimited exactly as are "single" records (C.7.2.4.1. above), but the delimiters must "line up" with the "|" delimiters of the first of the pair; i.e., they must be in the same columns. In both records of the pair, as in single records, the delimiters must be the first and last non-blank characters in their records. "Special" octets are represented by two hexadecimal digits in corresponding columns of the two records, with the upper "nibble" in the first (over) record, and the lower in the second. Also, because of the "lineup" requirement, the "horizontal" doubling, used in the single-record form to represent



an instance of the start/end delimiter in the data itself, can not be used; instead, "vertical" doubling (in which the character occupies the same column in both records of the pair) is employed. In all other cases, one of the pair of corresponding columns in the two records must carry the character to be represented, and the other must be blank.

To illustrate the method, let us assume that the string used in the ASCII examples above must be preceded by two octets: a parameter code (10110100 in binary, or "B4" in hex) plus a length indicator covering the string itself. This could be split into two "pieces", a "pair" and a "single", as follows:

```
|b1      ' 2   2 |
```

```
'4fThat's0allc &
```

```
&ladies & gentlemen!'
```

in which the comma and one of the blanks in the first "piece" has been "hexed". Note the vertical doubling of the apostrophe, which was the start/end delimiter used in this example.

#### C.7.2.4.3 Operator ASCII Data Input

This method of providing data for a /QUEUE-built event is interactive; the operator will be prompted for each "piece", and if that piece does not conform to the expected format, the software will immediately diagnose the violation and request reentry. The form itself has been adapted for operator convenience, but is essentially equivalent to continuation format.

First, with certain exceptions, "piece" delimiters are optional.

The data in each line entered are assumed to start in column 1, and end with the last non-blank character on the line. However, if the character in column 1, or the last non-blank character on the line, is a quote ("), apostrophe ('), ampersand (&), or vertical bar (|), it is assumed to be a delimiter and is excluded from the data itself.

If column 1, or the last non-blank character, of a line is other than one of the delimiters above, "implied" delimiters are assumed. This is as follows:

(1) On the first line entered, the left (start) delimiter implied is:

(a) a quote (") if the last non-blank on the line is a quote;

(b) an apostrophe (') otherwise;

(2) An ampersand (&) is implied in all other cases.

In other words, a "start" delimiter is supplied for the first "piece", and "continuation" delimiters thereafter. The operator must therefore explicitly terminate the data with a right delimiter on the last piece which matches the "start" delimiter (explicit or implied) on the first piece.

Except for the required column 1 start and the implied delimiters, all the ASCII continuation rules are in effect. Let us use the example for ASCII data continuation as it might be entered by an operator (lines are assumed to start at column 1):

That's

all, ladies & &

gentlemen!'

Several points should be noted in this example. First, the implied

AD-A173 442

DEVELOPMENT OF A VALIDATION SYSTEM FOR GROUP 4  
FACSIMILE EQUIPMENT(U) DELTA INFORMATION SYSTEMS INC  
HORSHAM PA NOV 85 NCS-TIB-85-8 DCA100-83-C-0047

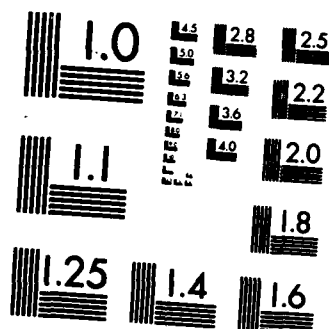
2/2

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

apostrophe, as start/end delimiter, requires its doubling on the first line and its use to terminate the data on the last line. Also, the explicit ampersand (&) continuation on the second line is needed to "protect" the text ampersand, which would otherwise be interpreted as the delimiter. Lastly, note the indent on the second line which represents a blank in the data itself.

#### C.7.2.4.4 Operator non-ASCII (over/under) Data Input

As opposed to operator input of plain ASCII data, this form requires explicit delimiters to satisfy the "lineup" requirements for a pair, and the necessity to mark the "over" record of a pair as such with the vertical bar (|) delimiters. This causes the form for operator input of "over/under" pairs to be the same as for continuation records, except that "left" delimiters must be in column 1.

To illustrate this, here is the example for "non-ASCII" continuation above as it might be entered by an operator (all lines assumed to start in column 1):

```
|bl      ' 2    2 |  
'4fThat's0allc &  
ladies and gentlemen!'
```

Note that, as in the data continuation example, "singles" and "pairs" can be mixed, as above, to provide the data for a single /QUEUE-built event.

### C.7.3 Provision of Information to Operator

The `"/NOTE"` command causes instructions or other commentary to be printed out for the operator's edification, and may optionally serve as a "breakpoint" in the performance of the test. The form of the `"/NOTE"` header is:

columns 1- 8: `/NOTE`

columns 9-16: `pauspec`

The effect of processing the `/NOTE` command is to echo following records, up to (but not including) the next record that has a slash (`/`) or dollar sign (`$`) in column 1 (the `/NOTE` header itself will have been output by the normal echoing of command headers). Following this printout (or screen display), the action taken depends on the `"pauspec"` parameter in columns 9-16.

The form of the `"pauspec"` parameter is either of:

`PAUSE`

`NOPAUSE`

The first form (`PAUSE`) results in suspension of processing, after the operator has been prompted to `"HIT RETURN TO PROCEED"`. If the operator simply enters a carriage return, processing of the test script will be resumed. Entry of a dollar sign in column 1, however, will cause control to be returned to the event queue (like the `PROCESS` parameter in a `/QUEUE` event; see C.7.2.1 above). If, in addition, the dollar sign is accompanied by a `"T"` or `"Q"` in column 2, it has the effect of a `"test quit"` or `"session quit"` command respectively.

The other form (`NOPAUSE`) causes test processing to continue without operator intervention. This is the default; blanks in columns

9-16 (or anything other than "PAUSE") has the same effect as "NOPAUSE".

#### C.7.4 Test Script Termination

The test "script" is terminated explicitly by the "/END" command, which has the following form:

```
columns 1- 8: /END
columns 9-16: prtspec
columns 17-24: prgspec
```

The effect of the /END command is to cause the test processing to "turn itself off"; i.e., the user-application layer interface module thereafter deletes the poll event which stimulates it to invoke the test processor, or otherwise sets itself to "ignore" it. Event queue processing, however, continues to its normal (or abnormal) conclusion.

The two parameters "prtspec" and "prgspec" control handling of the test log following completion of test performance. The first, "prtspec", has one of the following forms:

```
PRINT
<FORCED>
NOPRINT
```

The first and second ("**<FORCED>**") is generated by implied /END commands; see below) forms cause the entire contents of the log to be printed; the third form suppresses the printing. The default is NOPRINT.

The "prgspec" parameter may be either of:

```
PURGE
NOPURGE
```

The first form, "PURGE", causes the log to be purged, following its printing, if indicated; NOPURGE causes the log to remain "on file". NOPURGE is the default.

If a record with a dollar sign in column 1 is read from the test processing file, or end of file is encountered, an /END record of the following form is "implied":

/END    FORCED

which terminates test processing and causes the log to be printed but not purged.



END

12-86

DTIC